Mission Control Client for GENSO

Kristian Engh Lundgreen
Martin Kirch Dige
Thomas Paulin
Kasper Revsbech
Mikkel Gade Jensen
Kim Højgaard-Hansen

AALBORG UNIVERSITET

Department of Electronic Systems

Computer Engineering

Fredrik Bajers Vej 7C

Telephone +45 96 35 87 00

Fax +45 98 15 17 39

http://www.control.aau.dk

Title:

Mission Control Client for GENSO

Theme:

Complex distributed systems

Project period:

5th semester,
September - December 2006

Project group:

Computer Engineering, group 552

Participants:

Kasper Revsbech
Kim Højgaard-Hansen
Thomas Paulin
Martin Kirch Dige
Mikkel Gade Jensen
Kristian Engh Lundgreen

Supervisor:

Dan Bhanderi

Number of prints: 9

Number of pages: 118

Number of appendixes and character:

1 pcs. CD-ROM

Finished: December 21th, 2006

Synopsis:

This project covers the analysis and design of a Mission Control Client for the Global Educational Network for Satellite Operations (GENSO) project, a project managed by the International Space Education Board (ISEB). The main purpose of the GENSO project is to widen the communication window with student satellites. This is done by allowing remote control of ground stations connected in a network, The network interface application is called a Mission Control Client (MCC)

Initial requirements for the GENSO project, did not fulfill requirements for a sufficient requirement specification. Therefore work was done to restructure and reformulate these. The final requirement specification is still missing important parts regarding functionality, because the GENSO project is still in the requirement iteration phase.

Most of this project report documents the first design iteration from the requirement specification. The conclusion states that the design is ready to implement prototype code, but more iterations are needed before implementation can begin.

Preface

This project has been made by project group 552, Computer Engineering 5th semester at Aalborg University. The focus group for this report is people interested in distributed systems and satellite communication as well as the GENSO project under ISEB.

Figures and tables in this report are labeled with chapter and figure number e.g. 4.2 for 2nd figure in chapter 4. References to literature are shown as e.g. [8, p.25], meaning page 25 of source 8 in the bibliography. In appendix D on page 116 is a list of acronyms used throughout the report with acronym and expanded name. As an example the acronym GSS will be shown as Ground Station Server (GSS) the first time it is used, and as GSS subsequently. In appendix E on page 118 is a glossary of terms used in the report.

The project group would like to thank the following people for their involvement in the project:

- Supervisor Dan Bhanderi
- Abe Bonnema for his help on defining the requirements structure
- Neil Melville for his commitment as well as his constructive debates regarding the requirements structure

A CD-ROM is attached to the cover of the report. The CD-ROM contains the following:

- GENSO project documents
- WWW literature sources
- This report as PDF file

The GENSO collaboration effort

It has been chosen to explain some details of the project process, concerning the collaboration with the rest of the GENSO project. The GENSO project works across both different countries and different continents, and that of course puts constraints on the communication flow. At the same time the project objectives are aiming at building a global distributed system, in a way not done before. This sets hard requirements for the project management, to be able to control that everybody in the project moves forward in the same direction.

Project management in the GENSO project consist of a project leader, and a system engineering team. The system engineering team consists of a member from each assigned work package (see table 1.1 on page 11), but since no work packages have been assigned yet, the system engineering team is not fully functional. This has had an impact on the way the project has moved forward, specifically regarding the work on requirements. The project management has defined that the requirements has to be finished before the work packages can be assigned, even though this project group feels that the system engineering team should play an important role in this definition. The requirements for the GENSO project are confusing in their present state, since there is no link from the defined objectives to the actual system requirements. As described in section 6.2 on page 30 this project group and other participants have tried to begin the work needed for completing this link, but it has not been achieved yet.

One of the main reasons that the requirements provided by the GENSO project are not complete, are also due to the time, at which this project group entered the GENSO project. The project group has worked with the GENSO project one semester, but the GENSO project has just started to move into the design phase, and has been planned to last $2\frac{1}{2}$ years. This gives a non complete set of requirements, and too many questions that needs to be answered. Some of these loose ends has been filled by taking a choice, others are still in question. This of course results in a non complete solution.

Throughout the process there has been a lot of communication between the GENSO project and the project group. This communication has unfortunately mainly been through text communication, either via chats, through mails or through USENET. This way of communication is often a lot slower than just talking to each other, and unfortunately some misunderstandings are also inevitable. These difficulties has resulted in a significant waste of time, both for the GENSO project, but also for this project group, since significant parts of the work on the requirements has been discarded more than once.

The GENSO project has also held two workshops. The latest workshops was due when this semester started, and one member from this project group attended the workshop. The result of actually making sure that people from the whole project met each other face to face, and discussed how this can be done, is quite impressing, since most of the decisions actually made in the GENSO project, where decided at these workshops. A third workshop is in the planning phase, and the project group believes this will help the GENSO project move forward.

The whole process of working with a non complete set of requirements and trying to implement some parts of the GENSO project anyway, has not resulted in a finished solution, but the project group feels that this project report can be viewed as the way the whole project should be handled regarding requirements work. If the GENSO project does not define how to link the objectives with the requirements, the risk of not ending up with a working system will be significant. This project report does point out a lot of the holes in the design and requirements as they are stated today, and the project group hopes that the GENSO project management will try to find a way to handle this.

Kim Højgaard-Hansen	Mikkel Gade Jensen
Martin Kirch Dige	Thomas Paulin
Kasper Revsbech	Kristian Engh Lundgreen

Contents

1 Introduction

Satellites have existed since late 1950's when the Soviet Union launched the first satellite, Sputnik 1, into orbit. Today there are approximately 2500 satellites in orbit around the Earth[15]. While the majority of the satellites are for commercial or military use, there are also several satellites in orbit for educational purposes, as several universities around the world have developed their own satellites. Communication between the satellite and Earth is accomplished by means of a ground station usually placed at the university where a given satellite was developed.

One issue when working with satellites has always been the communication between the ground station and the satellite (see figure 1.1). Communication is only possible when the satellite passes over the ground station which means that the time slot in which communication is possible, is narrow. Student satellites typically has a round trip time around the Earth of approximately 90 minuttes, and a pass duration of 10 minuttes at best [11, p. 4].

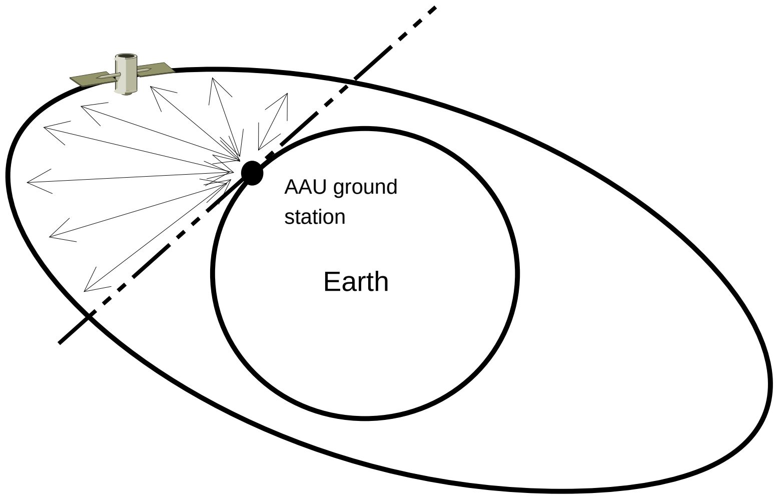


Figure 1.1: The figure shows how communications is only possible when the satellite passes over the ground station. The proportions of the figure are not exact.

The limited amount of time, where the satellite is available for communication, puts a lot of constraints on when data can be exchanged between the ground station and the satellite. If it were possible to use several ground stations around the world, the satellite would be available for communication much more often.

1.1 GENSO

Global Educational Network for Satellite Operations (GENSO) is a project that has been started by the European Space Agency (ESA) Education Department under the auspices of International Space Education Board (ISEB)[10, p.4]. The project aims at creating better communication possibilities for student satellites at a global level. As explained in section 1 on the previous page the communication with a student satellite is usually limited to only a single ground station. The GENSO project aims at creating a global network of collaborating ground stations, which can all be used by participating satellites. The project is defined by the following objectives[11, p.6]:

- Unparalleled near-global levels of access to educational spacecraft in orbit.
- Optimised uplink fidelity by calculation of real-time link budgets and automatic uplink station selection.
- Remote control of all participating ground stations.
- Remote access for operators to real-time mission data, even in cases when their local ground station is experiencing technical difficulties.
- Downlink error-correction by comparing multiple data streams.
- A global standard for educational ground segment software.
- An optional well-defined standard solution for educational ground-segment hardware (in order to expedite participation in GENSO).
- An optional well-defined standard design solution for educational space segment communications hardware (in order to expedite participation in GENSO).
- Support a common interface for applying for frequency allocation and coordination.

These objectives form the overall goals of the whole project. The objectives are in the process of being further developed into a requirement document, defining what should be developed to make this possible. The document defining the initial draft of requirements, will from here on be referenced as the *list of ideas* (see section 6.1 on page 30). The *list of ideas* can be found on the CD-ROM.

The GENSO project was initiated by ESA Education Department as an assessment study in June 2006. A group consisting of The Radio Amateur Satellite Corporation (AMSAT), University Space Engineering Consortium (UNISEC), Student Space Exploration and Technology Initiative (SSETI) and several universities including Aalborg University (AAU), were asked to determine the technical feasibility of the project and define a comprehensive set of technical requirements with suggested design solutions. This work resulted in the definition of a set of work packages. These work packages are distributed to the participating organizations, which will then carry out the needed work. The contents of each work package is described in Table 1.1 on page 11. The GENSO project development phase is scheduled to last around $2\frac{1}{2}$ years from june 2006 to November 2008[10].

The work packages are formed from a "Conceptual Layout"[11] of the system, as shown in figure 1.2:

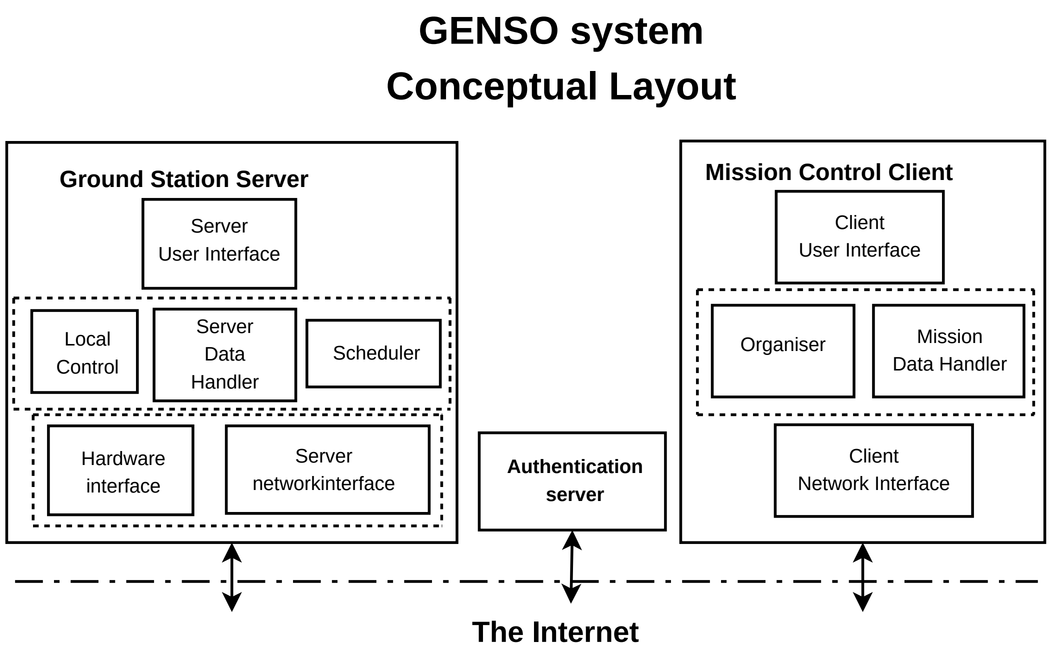


Figure 1.2: The figure shows the "Conceptual Layout" of the GENSO system. The system is divided in three nodes; The Mission Control Client, the Ground Station Server and the Authentication Server. The first two are further split up into logical modules. These modules forms the borders of the different work packages. The figure is a simplified version of the original "Conceptual Layout" figure without communication lines[11]. The layers are just logical splits, not completely determined.

The three nodes are described as follows:

Ground Station Server (GSS)

Quote:
An instance of the GSS should be running on the controlling computer(s) of each hardware ground station in the network. This software consists of a Hardware Interface module to control the local hardware (a large library of drivers would be distributed with it, along with specifications for users to develop new drivers), a Local Control module to control the Hardware Interface, a Scheduler module to plan and execute passes, and a Network Interface module for communication with other

instances of the GSS and MCC.[11, p.7]

Mission Control Client (MCC)

Quote:
Each spacecraft project should run one instance of the MCC on the computer used by the spacecraft operators. This software consists of a Graphical User Interface, a Data Handling module for storing and retrieving mission data, an organiser for booking uplink sessions and retrieving downlink data, and a Network Interface module for communication with instance of the GSS.[11, p.7]

Identification and Authentication Server (quote)

Quote:
Users of both the GSS and the MCC must login to identify themselves, using a central authentication server (or mirrors of it) for identification and authentication. From these logins the server generates two dynamic lists and distributes them throughout the network:

- *The Ground Station Servers List (GSSL), being a list of all instances of the GSS registered in the network, and their various details.*
- *The Participating Spacecraft List (PSL), being a list of all instances of the participating spacecraft registered in the network, and their various details, including those of the appropriate instance of the MCC.*

[11, p.7-8]

Each software development work package will be assigned to a development team, who will be responsible for the design, development and implementation of the work package content. There are also other types of work packages for non software development. The work packages are defined in Table 1.1 on the next page.

Work packages for the GENSO system	
A	Hardware Interface and Local Control modules of the Ground Station Server, for Microsoft Windows.
B	Hardware Interface and Local Control modules of the Ground Station Server, for Linux.
C	Scheduler, Data Handler and User Interface modules for the Ground Station Server (OS independent).
D	Organizer, Data Handler and User Interface modules for the Mission Control Client.
E	Authentication Servers and the Network Interface modules for both the Ground Station Server and the Mission Control Client.
F	Standard hardware ground station in the network, and coordinated development of a library of appropriate device drivers.
G	Defining, developing and controlling all interface specifications, tracing and maintaining the requirements tree, coordinating functional testing plans and progression, and taking project-level technical decisions.
H	Maintenance of appropriate infrastructure to support the project, including IRC, NNTP, FTP and HTTP servers.
I	Investigating relevant legal issues, developing user agreements, identifying, recruiting, and coordinating network participants, and design and content of the project website

Table 1.1: The GENSO work packages

This is the information used throughout the report as source material about the GENSO project.

1.2 Report structure

This section contains an explanation of the structure of the report and gives a brief introduction to each part in the report.

The report is split into the following parts:

I Analysis

II System Description

III Design

Analysis

The analysis contains a description of what a ground station is, the AAU/Svalbard setup and a description of the existing ground station networks:

- Federal Ground Station Network
- Ground Station Management Service

System description

This part primarily concerns the work with the requirements to the system. The part begins with a use case analysis of the MCC followed by an explanation of the requirement iteration process that has been made for both the GENSO project, and the for this project. This ends up whit the requirement specification of the MCC.

Design

The design part describes the overall design of the MCC and the overall design methods used. This is followed by a description of each module in the MCC. The design is followed up by a design conclusion. The design part is ended with the conclusion of the project and the perspectives.

Appendix

Appendix A contains an explanation of satellite trajectories to understand how to track satellites, and in general how to evaluate the paths of satellites.

Appendix B Contains a description of the Two-Line Element (TLE) provided by North American Aerospace Defense Command (NORAD) to understand how to interface to the Predict library.

Appendix C Contains the accepttest specification describing the test cases constructed to test if the requirements in the requirement specification is fulfilled.

Appendix D Contains the acronyms used throughout the report.

I Analysis

2 Existing solutions

Although the GENSO-project has defined which work packages that must be developed and what they consist of, a general analysis of the initial problem is examined. In order to define an adequate problem formulation, there are aspects of the way present satellite systems work that need to be analyzed. In the analysis an overview of which conditions needs to be fulfilled to make communication between a satellite and the ground station of a university possible, will be given. These conditions will be based on the configuration at AAU, and reflects over the advantages and disadvantages with the current setup. After this, two other projects concerned with network based ground stations will be described. The first project to be described is the Federal Ground Station Network (FGN) project, which will be followed by a description of the Ground Station Management Service (GSMS) project. Both projects will be analyzed to get an overview of the requirements needed by such a collaboration of ground stations as well as design considerations.

The results of the analysis will then be used to construct the problem formulation of this project.

2.1 Ground station in general

A significant part of building a satellite is to establish a two-way communication link from the Earth to the satellite. Two-way communications is needed for requesting specific data from the satellite and evaluate the house keeping data on the satellite platform. The basic link to the satellite is given through a radio connection, which is initiated by a ground station.

A ground station is a set of equipment, needed for providing control, data and tracking mechanisms for satellite spacecrafts in orbit around the Earth. The following basic hardware is needed to construct a ground station:

- Antenna, for distributing radio signals
- VHF/UHF radio, for transmitting and receiving modulated signals
- Antenna rotator, for tracking systems
- Preamplifier, needed to amplify the received signals
- Modem, for modulating and demodulating the radio signals
- PC or controller, for controlling and timing the hardware

The tracking system of a ground station is an important part, since the satellite is only visible by the ground station in a short period of time, and the received signal is weak. Therefore it is necessary to make the antenna omnidirectional. As the satellite moves across the sky, the ground stations has to compensate by moving the antenna or it will loose the communication link to the satellite. If the antenna is not in the right position, the signal quality will fall, and the signal maybe hard to demodulate due to interference.

The modem inside the ground station setup has to match the modulation profile chosen by the satellite developer. There are many different kinds of modulation profiles made for radio communication. Therefore it is necessary to know the profile of the satellite before the session is started. Normally these modems are implemented in hardware, and accessed by a standard RS232 interface.

The PC or controller is mostly a special implementation of some satellite specific software, but basically only drivers for radio and antenna rotator is needed.

2.2 AAU/Svalbard ground station setup

AAU has constructed a ground station to communicate with student satellites. Originally it was build to communicate with AAUs first satelite AAU CubeSat. The ground station has since been modified to fit the communication protocol and modulation of the SSETI Express[1], developed by the European Space Agency (ESA). In the future the AAU ground station will be modified to support communication with the AAUSAT II.

The AAU ground station is using an ICOM VHF/UHF radio, with S-band options. In connection with the radio there is a TNC7 modem, which supports modulating and demodulating Fast Frequency Shift Keying / Minimum Shift Keying (FFSK/MSK) signals. AAU ground station is using the UHF band for uplink data, and S-band for downlink. This decision gives a faster downlink, than uplink.

Attending the SSETI Express project AAU redeveloped a ground station, and a Mission Control Center[2]. In order to ensure a backup ground station, and a larger communication window, AAU was allowed to use a ground station in Svalbard. See figure 2.1 on the following page

[1]http://www.esa.int/SPECIALS/sseti_'express/index.html

[2]Must not be confused with the Mission Control Client that the GENSO network defines. The Mission Control Center are the direct link to the satellite.

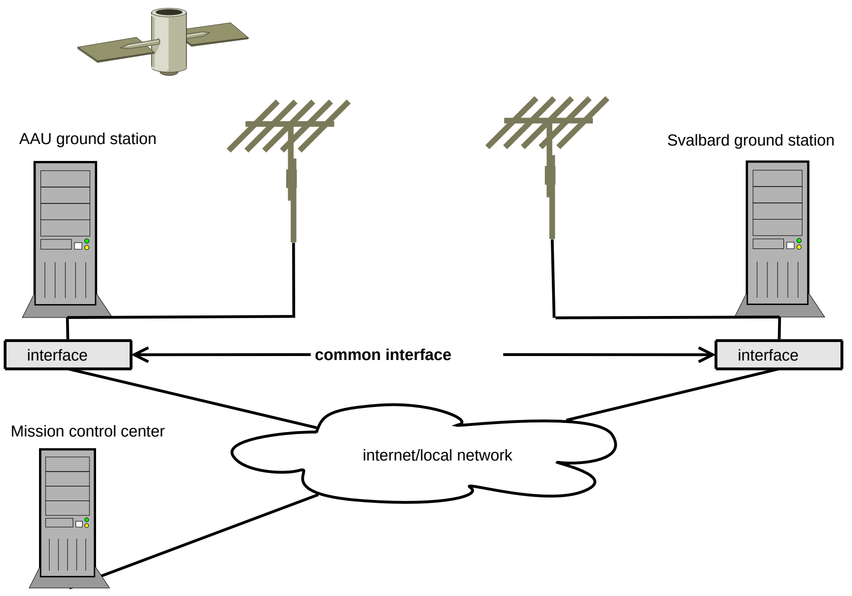


Figure 2.1: An overview of the two ground stationes located in Svalbard and AAU. The two ground stations have the same driver interface therefore the Mission Control Center can send the same command regardless of which ground station it is communicating with

As shown in figure 2.1 the AAU solution contains the following two parts:

- Mission Control Center
- Ground Station Server

The two systems are split into several subsystems which are further documented in their respective project reports (see [2] and [9]) The main concern in this project is the handling of the communication between two or more ground stations and the design of the Mission Control Center allows usage of several ground stations through a GUI providing facilities for sending commands and receiving mission data. Thus it is designed to be fully controlled, by the user, to select which ground station to connect to.

The communication to the ground station is handled in the same way regardless of which ground station it is connected to. This is ensured by applying the same interface to the two ground stations. In order to get the same interface the ground station server in Svalbard is running the same software as the server on AAU, the only thing changed is another set of drivers to handle the different hardware setup. The drives then provide the same common interface to the rest of the software as in AAU.

The AAU/Svalbard setup is only a semi distributed system. The user has to manually choose between two servers, which leaves the user with an extra workload. It would be preferred for the user not to worry which ground station is being used for a communication session as long as the connection to the satellite is active.

2.2.1 Summary

Todays configuration at AAU consists of a complete system with a few backup ground stations that can be contacted if the main ground station fails. This configuration gives possibility of controlling and adjusting anything at the local ground station, but the risk of not being able to communicate with the satellite because of a broken ground station is high. Another concern is that communication with the satellite is not possible whenever the satellite is out of view of the ground station.

2.3 Existing networks of Ground Stations

Now that the current set-up of the AAU/Svalbard ground stations has been described, the next step is to analyze already developed networks of ground stations. As this is not an entirely new idea, and GENSO is not the first initiative to develop such a network, it is possible to analyze the FGN and the GSMS projects. which are similar to the GENSO project.

2.3.1 Federated Ground Station Network

In a attempt of creating a collective communication satellite system National Aeronautics and Space Administration (NASA) and Standford University developed guidelines for how such a system should be engineered. The FGN is an effort to standardize the configuration of ground stations world wide, so they can take part in a global ground station network. The FGN provides specifications on the interface the ground station must respect in order to become a part of the network, this is called the Ground Station Markup Language (GSML).

GSML is an Extensible Markup Language (XML)-based language constructed by Stanford university to act as an Application Programming Interface (API) to the FGN. It provides a hierarchical command and control structure with low level device commanding of the ground station hardware, and high level mission planning functions.

By creating this network the goal is to obtain the following advantages [7]:

- Reducing mission operations cost
- Increasing mission yield and capabilities
- Provide baseline satellite infrastructure to make science data more accessible and enable experiments to have wider impact

- Make this infrastructure robust and affordable

As the above mentioned are only system specifications Stanford University developed the Mercury Ground Station System (MercuryGS), being an actual software solution following the GSML.

Mercury Ground Station

The MercuryGS system is a software suite designed to allow remote control and commanding of ground stations via the Internet.

The heart of MercuryGS is a web frontend software, a database, and the ground station manager (GSM). The web frontend provides a basic, complete command and control interface to all ground station services. Through the web interface, you are able to configure the ground station, control hardware, and schedule ground station resources. The database provides storage for all this information. The GSM is an independent application. It polls the database for active sessions and spawns the software to enable session execution.

Summary

The MercuryGS enables remote access to ground stations, expanding the possibilities of communicating with the satellites, by enabling access to ground stations all over the world. The system works, and has been implemented and in use with a range of student satelittes. The systems uses well known and well proven software to accomplish most of the functionality, and a modular design to ease implementation. A ground station specification has also been developed, together with a generic API called GSML, to ease the implementation of both new ground stations and remote use of these [13].

The MercuryGS is not a network of ground stations. This means that it is only meant for controlling one ground station at a time, and scheduling and control for more than one has not been planned nor designed.

2.3.2 Ground Station Management Service

The GSMS was originally started in 2003 by 5 universities[3] in Japan and today there are 15 participants in the project. The main objective of the project is to develop an internet-based open source autonomous tracking system. Like the FGN, GSMS focuses on the definition of the interfaces and software design rather than on the actual software development. GSMS has already been used with several satellites and there is still ongoing development of the project.

[3]Universities from Tokyo, Nihon and Kyushu as well as the two Institutes of Technology of Tokyo and Hokkaido

GSMS overview

In contrast to the FGN, GSMS has introduced a further major part into the network, that is the Central Server (CES). Thereby the overall GSMS system consist of the following parts: (see figure 2.2).

- Client: A GUI interface.
- Operation Server (OPS) : Located at the ground station and responsible for the communication with the satellite.
- CES: A central server responsible for handling authentication of the users of the system. Furthermore the CES is responsible for the large scale calculation of the scheduling.

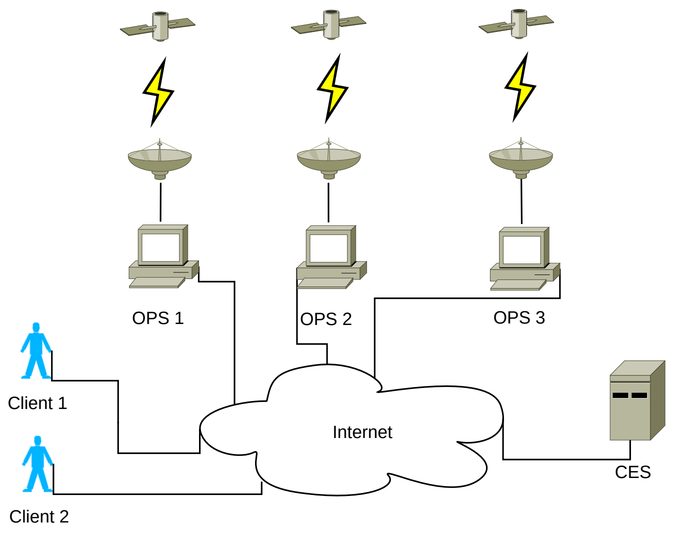


Figure 2.2: The figure shows an overview og the GSMS system. In this case there are 2 clients, 1 CES and 3 OPS all connected through the internet.

The main aspect of the GSMS it the definition of the interfaces. These are used as guidelines for how the different software parts are interconnected. Because the focus of GSMS is not the development of the software, but defining how the communication and flow should be, it is important with clearly defined interfaces. Thereby it is possible to exchange any software part or change software modules like drivers without redesigning the whole system.

The CES takes care of a lot of work, that is necessary for the GSMS to be operational. This work concerns both authentication as well as the scheduling of passes. This can result in reduced performance, as the scheduling procedure can take up much of the CES's time. In case of the network grows, this becomes a hindrance. As to authentication, it is necessary to have some centralized system, so a possible solution could be to divide these two tasks into two separate systems.

2.4 Analysis conclusion

The GENSO system generally goes one step further than previously developed systems. First of all it takes the cooperating ground stations to a global level, where ISEB defines the scope. This means that instead of NASA having Mercury, Japan Aerospace Exploration Agency (JAXA) having GSMS etc. all these organizations join up in a common system. This means that the actual reason for making the GENSO project is more political than based on new functionality.

The two previously developed systems described here will be used as knowledge base for developing GENSO. Mercury has constructed a general but also flexible software suite, allowing remote control of another ground station. MercuryGS does not provide scheduling of more than one pass over ground stations. Mercury uses XML to communicate between nodes in the network, allowing a wider use of programming languages to develop new software. GSMS actually achieves most of what GENSO wants to achieve, but are having scalability difficulties with the chosen design.

There is also a new objectives in the GENSO project. The objective concerning using more than one ground station at the same time, and using this to achieve a better level of error correction.

3 Problem formulation

The preliminary analysis has lead to the problem that this project wishes to solve. To make it possible to expand the communication window with a satellite, the use of more than one ground station is desired. This can be achieved in more than one way, as seen in the description of both the Mercury system, and GSMS. The aim of GENSO is to create a similar system, but at a larger scale and with extended functionality. GENSO want for the participants to be responsible for one or more work packages, but the work packages were still in the "decision process" at the start of the semester. Thus instead of choosing one work package, the group has chosen to work with the development of the whole MCC

The problem this project wishes to solve is:

How can a Mission Control Client (MCC) of a distributed network of ground stations be developed to suit the GENSO project?

The MCC is defined as a control interface of the GENSO network. The focus of the project will includes the following subjects:

- Definition of use cases and requirement specification
- Design, implementation and test of the MCC

II System description

4 Use case analysis

The GENSO project does not implement a use case analysis to describe the main functionalities of the system. This project will implement one for the same reasons. A use case analysis is used to capture the requirements from the customer[5] and is typically the first activity in object oriented software development. This use case analysis is constructed by reverse engineering the GENSO *list of ideas*. Figure 4.1 on the next page shows the use case diagram of the MCC with associated actors:

- An **Operator**, whose main objective is to control and schedule the mission of a satellite
- **Predict**, which is an application for predicting satellite trajectories
- An **Authentication Server (AS)**, which is responsible for management of the network. It holds the Ground Station Server List (GSSL) and the Participating Satellite List (PSL).
- A **Ground Station Server (GSS)**, which is the gateway to the satellite, responsible for radio link communication
- A **Satellite**

In the following sections, each captured use case in the MCC will be described with regards to objective, flow and exceptions.

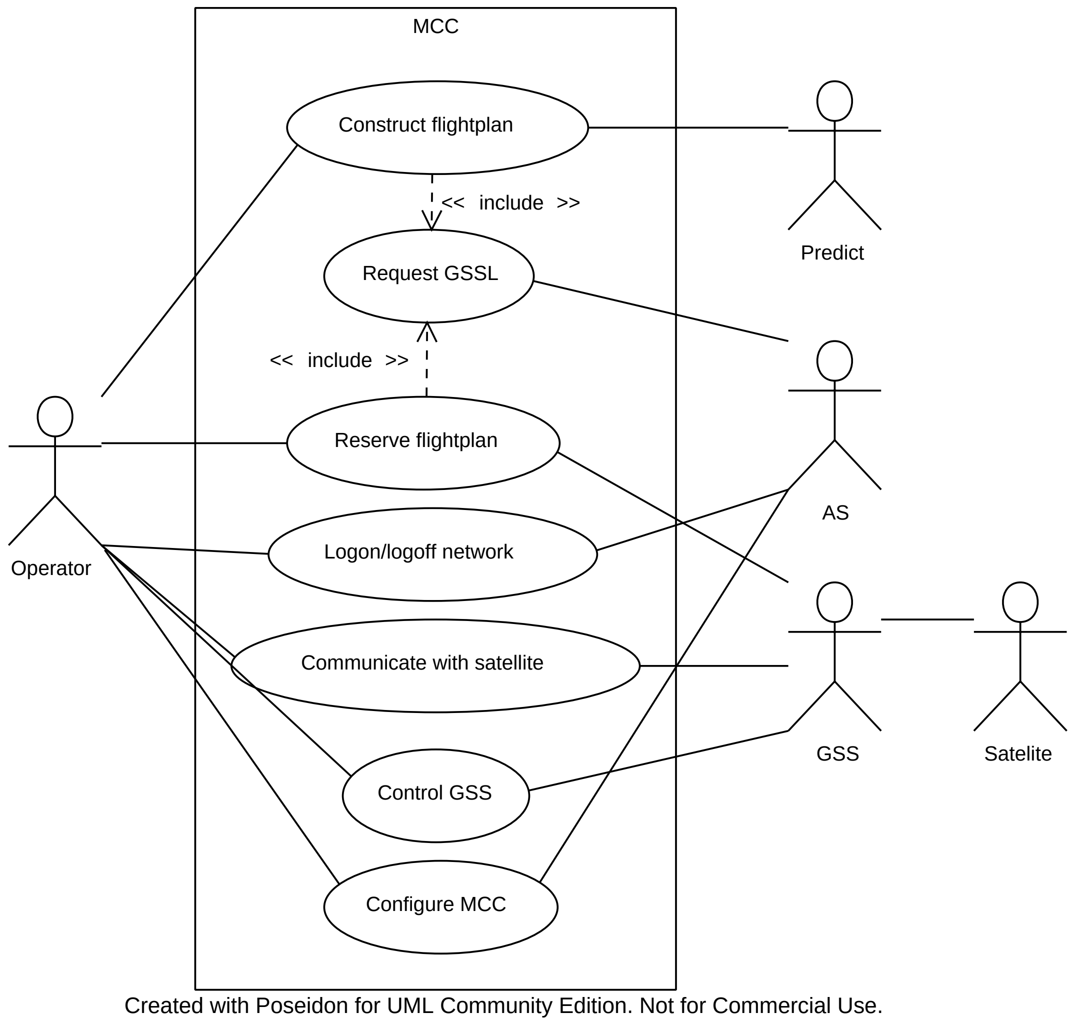


Figure 4.1: The Figure shows the MCC use case diagram. The relationsship notation between the actors GSS and satellite is not a part of the UML standard, but it visualizes the role of the GSS being a gateway for communicating with the satellite

Construct flight plan

Objective

The flight plan is a chronological list of the ground stations that the satellite passes over in a given period of time. It must also contain pass over time for each ground station. The objective is to calculate which ground stations are in range of the satellite and available in the time period specified by the operator and then show it to him.

Flow

1. The operator tells the MCC in which time span he wants to communicate with his satellite
2. The MCC uses Predict to calculate the passover route
3. A list of the available GSSes are requested from the AS through the included use case "Request GSSL"
4. The MCC uses the passover route and the GSSL to determine the ground stations in range
5. The sequence of ground stations is shown to the operator

Exceptions

- If predict is not responding, display the exception to the operator
- If the AS is not responding, display the exception to the operator
- If no ground stations are in range or available, display it to the operator

Reserve flightplan

Objective
When the flight plan is constructed, the involved ground stations must be reserved for satellite uplink/downlink communication.

Flow

1. The operator chooses which GSS to put into the flight plan and reserves it
2. The MCC requests the necessary time slots from the involved GSSes
3. Every GSS reports to the MCC if the requested time slot is granted
4. The results are displayed to the operator

Exceptions

- If a GSS is not responding, request GSSL again and try later.
- If the AS is not responding, display the exception to the operator

Logon/logoff network

Objective
To use the functionality of the network, the MCC must logon first.

Flow

1. The operator wants to use the network
2. The operator supplies the MCC with logon
3. The MCC is authenticated by the AS
4. The operator can use the network
5. When the operator is done using the network, he logs off the network

Exceptions

- If the MCC can not be authenticated, the message "Wrong logon name or password" must be displayed to the opearator.
- If the MCC can not connect to the AS, a timeout message must be displayed to the opearator.

Communicate with satellite

Objective
Communication from the MCC to the satellite is possible when a GSS time slot is available. The MCC connects to the GSS and then the MCC can use the GSS to send commands and download data from the satellite. All uplink and downlink data must be saved in a database at the MCC for later viewing. The MCC UI must provide information about the connection status to the current GSS.

Flow

1. The MCC Initiates the connection to the regarding GSS
2. The GSS acts like a data gateway
3. The MCC transmits commands to the satellite using the GSS
4. The GSS forwards downlink data from the satellite to the MCC
5. At the end of the session a pass report is generated by the GSS and requested by the MCC to display to the operator

Exceptions

- If no more GSSes are available in the session a message must be shown to the operator

Control GSS

Objective
It must be possible for the operator to fine tune specific parameters at the GSS concerning the communication session, such as Doppler correction and antenna position. This must be done in both a manual and a semi automatic way. The GSS will by default calculate these parameters, but the need of correction may appear.

Flow

1. The MCC provides the operator with the settings of the GSS
2. The operator observes data corruption of the downlink/uplink stream.
3. The operator changes a number of the GSS settings
4. The data downlink/uplink stream is corrected

Exceptions

- If the error can not be corrected by changing the GSS settings, the error must be located elsewhere.

Configure MCC

Objective
The operator needs to specify which satellite is attached to the MCC and a number of other MCC settings. This use case also handles the specification of satellite parameters in the PSL, such as satellite name, modulation and frequency.

Flow

1. The operator chooses which setting to change
2. The MCC changes the setting

Exceptions

- If changes to the PSL has been made, the new settings must be sent to the AS

5 Functionality delimitation

In the work constructing the use case analysis by reverse engineering the *list of ideas* described in section 6.1 on page 30, it has been chosen to delimit this project from some of the functionality. The functionality will be described here as text instead, to make it obvious that it has been left out of the design, not forgotten in the reverse engineering work.

Distinguish between uplink passes and downlink passes
The *list of ideas* states that the GENSO system shall be implemented with functionality to distinguish between uplink and downlink passes. This should be done to let the GSSes react autonomous with satellites that passes over them. The GSSes could then download different mission data for later retrival by the MCC. This functionality has been left out since it is not considered important in an initial version of the GENSO system.

Multiple downlink
The *list of ideas* states that the GENSO system should be able to allow a satellite operator to use more than one GSS at the time. This should be possible because multiple downlink streams could be compared for better error correction. It has been chosen to leave this out in this project since it is not considered relevant in an initial version of the GENSO system.

Quality factors
The *list of ideas* states that the GENSO system shall be implemented with a quality factor to weigh nodes in the network against each other. This should be done to give the operator a better view of what nodes would be best to book. It has been chosen to leave out this functionality because it is undefined how this quality factor should work.

Scheduling Lottery
The *list of ideas* states that the GENSO system shall be implemented with a special way of deciding which MCC is granted a pass when multiple MCCs tries to book a pass. All MCCs should be able to make initial bookings 24 hours before a pass. 6 hours before the pass, a lottery is weighing the different booking requests against each other should decide which MCC is granted the pass. It has been chosen to design the system with a "First come, first serve" scheduling algorithm instead. This can work in an initial version of the system.

Integrated Satellite application
The *list of ideas* states that the GENSO system shall be implemented with an MCC that is able to send commands to a satellite. Since this requires a definition of a GENSO satellite protocol, and this work has not been done, this functionality has been left out. Instead the system will be designed in such a way that it acts as a communication interface to the satellite. The actual commands to the satellite must be send from a application separate from the GENSO system,

called “Satellite Application”.

Overlapping passes
The *list of ideas* states that the GENSO system shall be implemented with a protection mechanism, securing that no overlapping passes can be booked. This is done to secure that two GSSes cannot send uplink commands to the same satellite. It has been chosen to leave out this functionality in this project, and let it be up to the satellite operator to make sure that doesn't happen.

Now that the delimitation of functionality has been described, the requirement work will be described.

6 Requirement iteration process

In every software / hardware project there must be a requirements specification. This also applies to the GENSO project. Because the project group joined the GENSO project in an early phase the requirements work was still ongoing and the group was asked to contribute on the requirement work. In this chapter the outcome of that process will be described.

6.1 Existing requirement structure

The initial implementation of the GENSO project requirement specification was a long, numbered list of requirements. It was formed from multiple discussions of the project and the ideas of the participants. This list is referred to as the *list of ideas* because the requirements is written as ideas and wishes to the system not as specific and testable requirements. The **list of ideas** can be found on the CD-ROM.

The structure of the *list of ideas* was multiple levels of indentation as a result of new ideas being added and spawned by other ideas. It had no classification of the individual requirements which made it hard to gain an overview of the entire system. Further more there was not a sufficient definition of the requirements defined by the actors in the system.

For this reason it was necessary to restructure and redefine the requirements from the *list of ideas* into a new structure, providing a better overview and traceability between the different levels of requirements. This project group and Abe Bonnema (System Engineer at Delfi C3) from TU Delft was assigned to develop a new structure and guidelines of how to plot requirements.

6.2 The new structure

In order to achieve a more sufficient structure Abe Bonnema proposed to use the structure in Figure 6.1 on the next page. On the figure the name G-Net is used as it was made before the project was named GENSO.

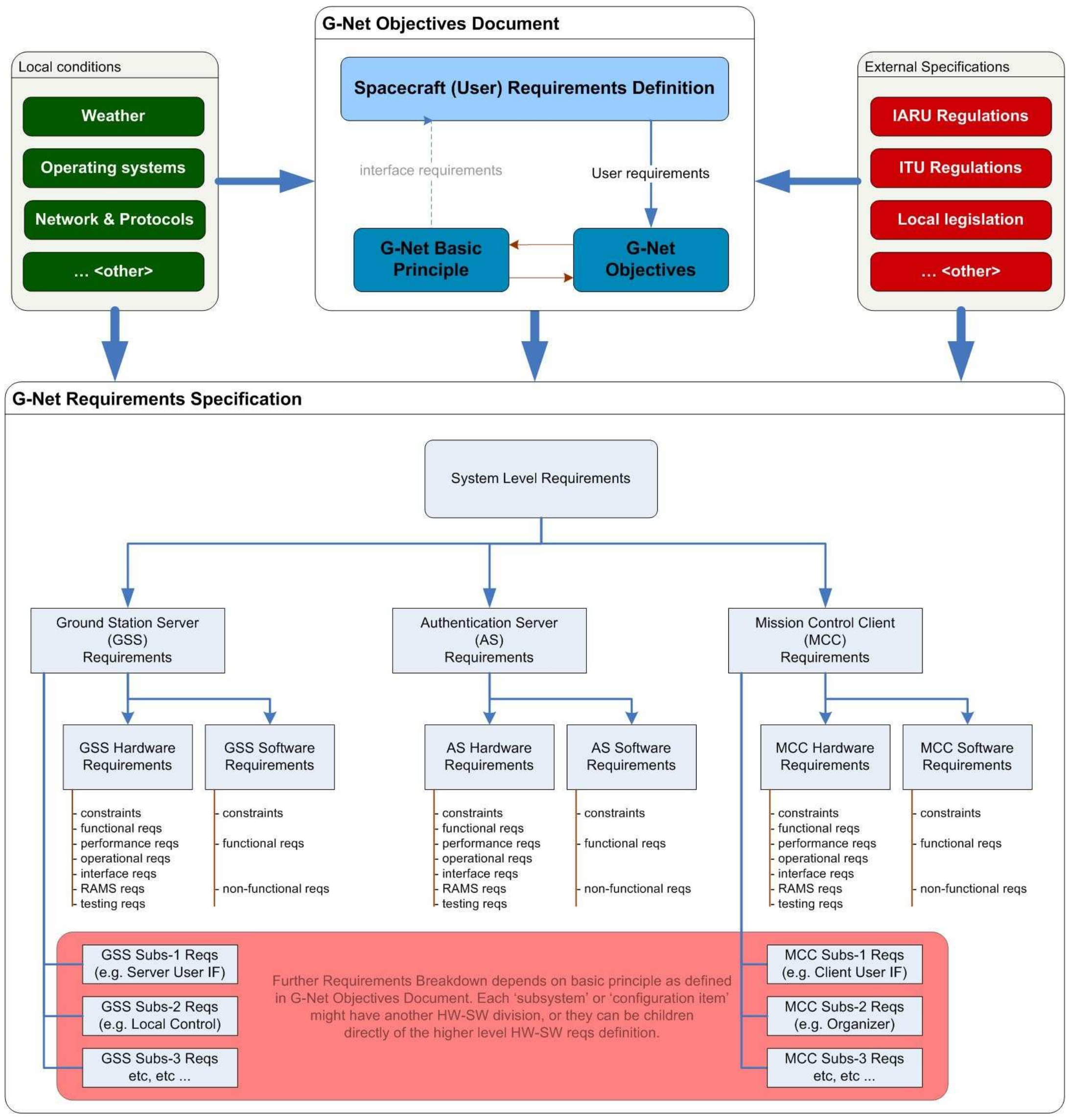

G-Net Objectives Document
Spacecraft (User) Requirements Definition
interface requirements
User requirements
G-Net Basic Principle
G-Net Objectives
Local conditions
Weather
Operating systems
Network & Protocols
... <other>
External Specifications
IARU Regulations
ITU Regulations
Local legislation
... <other>
G-Net Requirements Specification
System Level Requirements
Ground Station Server (GSS) Requirements
Authentication Server (AS) Requirements
Mission Control Client (MCC) Requirements
GSS Hardware Requirements
GSS Software Requirements
AS Hardware Requirements
AS Software Requirements
MCC Hardware Requirements
MCC Software Requirements
constraints
functional reqs
performance reqs
operational reqs
interface reqs
RAMS reqs
testing reqs
non-functional reqs
GSS Subs-1 Reqs (e.g. Server User IF)
GSS Subs-2 Reqs (e.g. Local Control)
GSS Subs-3 Reqs etc, etc ...
MCC Subs-1 Reqs (e.g. Client User IF)
MCC Subs-2 Reqs (e.g. Organizer)
MCC Subs-3 Reqs etc, etc ...
Further Requirements Breakdown depends on basic principle as defined in G-Net Objectives Document. Each 'subsystem' or 'configuration item' might have another HW-SW division, or they can be children directly of the higher level HW-SW reqs definition.

Figure 6.1: The figure shows the requirement structure proposed by Abe Bonnema. This proposal requires that user requirements and mission objectives forming the *G-Net objective document* are defined before the actual work on the requirements begins. In *System level requirements* the *G-Net objective document* requirements are derived into requirements defining how the system fulfill them. The system is then split into the three systems: GSS, AS and MCC. And the *system level requirements* are then derived into the respective system(s) defining how to fulfill them. If there is a need of further division there can be defined sub-systems to each system, and derive the system requirements into the subsystems

Given the structure defined in Figure 6.1 a system had to be build containing the possibility to make requirements and cooperate from different locations. To do this job the group proposed to use a DokuWiki [1] system. A DokuWiki is a website that allows users to edit and share pages with each other through a web browser. It is organized with a hierarchical structure of the pages which provides an easy navigation to the viewer/editor. This means that it is effective for collaborative editing and sharing of documents. There are many different kinds and layouts of a wiki system, but for this project the DokuWiki has been chosen because it is targeted at creating documentation for developer teams. It has a simple layout, is PHP powered and stores everything as text files which do not require a database from the hosting server. [4]

6.3 Wiki structure

The requirement structure was implemented in the DokuWiki as an abstract structure with one or more pages at each level. This means that the navigation is accomplished by clicking through the levels of the structure. The levels is implemented as described in Figure 6.1. This lead into to the levels:

System level

- GSS
- AS
- MCC

The requirements in the regarding system is then further categorized into the two structures:

Hardware

- Constraints
- Functional requirements
- Performance requirements
- Interface requirements
- Testing requirements
- Reliability, Availability, Maintainability and Safety (RAMS) requirements

Software

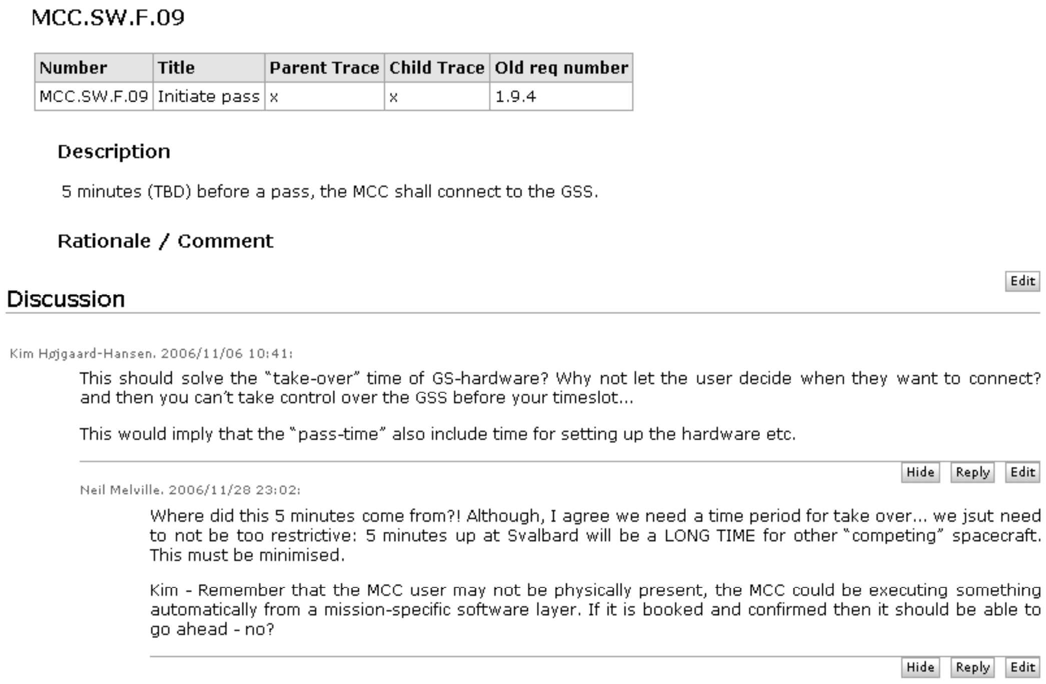
- Constraints
- Functional requirements

[1] http://wiki.splitbrain.org/wiki:dokuwiki

- non-functional requirements

Each requirement was added as a separate page in the wiki which makes it possible to add a discussion about the purpose and phrasing of it, and to use revision control. It is also possible to organize the requirements by creating pages with an overview linking to the desired requirement pages. The table in Figure 6.2 is used in each requirement page to achieve traceability and to specify number and title. This makes every requirement traceable from a User level requirements to specific requirements. I.e. each requirement page contains:

- A tracing table
- A discussion block
- The requirement itself



MCC.SW.F.09

Number	Title	Parent Trace	Child Trace	Old req number
MCC.SW.F.09	Initiate pass	x	x	1.9.4

Description

5 minutes (TBD) before a pass, the MCC shall connect to the GSS.

Rationale / Comment

Discussion

Kim Højgaard-Hansen, 2006/11/06 10:41:

This should solve the "take-over" time of GS-hardware? Why not let the user decide when they want to connect? and then you can't take control over the GSS before your timeslot...

This would imply that the "pass-time" also include time for setting up the hardware etc.

Neil Melville, 2006/11/28 23:02:

Where did this 5 minutes come from?! Although, I agree we need a time period for take over... we jsut need to not be too restrictive: 5 minutes up at Svalbard will be a LONG TIME for other "competing" spacecraft. This must be minimised.

Kim - Remember that the MCC user may not be physically present, the MCC could be executing something automatically from a mission-specific software layer. If it is booked and confirmed then it should be able to go ahead - no?

Figure 6.2: An example of a requirement page in the wiki system with tracing table, comments and a discussion.

Like the example in Figure 6.2 each requirement is numbered with a dot notation referring to the location of the requirement. E.g. MCC.SW.F.09 for a functional software requirement in the MCC. In general: $< system > . < SW/HW > . < type > . < req.\ number >$. The Rational/Comment section is for various purposes e.g. descriptions of changes in the requirement or to-do jobs.

6.4 Filling in the new structure

Is was decided that the responsible team of each work package should move the requirements from the *list of ideas* into the requirement structure, but a problem that was appointed by the group and by Abe Bonnema was that the user requirements and the system requirements was not defined or derived in the *list of ideas*. This means that it is hard to trace the impact on changing a requirement in e.g. the MCC level.

Furthermore it has been exhibited that there is a difference in the way that the participants in the project understand the work on requirements. The proposal from the group and Abe Bonnema suggest that requirements is defined in a strict way e.g. as defined in [1]. It is also suggested that there shall be a traceable flow down between the different levels in the system. To achieve a traceable flow down and a consistent requirement tree the approach should be that the system engineering team should work out the top levels and secure that all that requirements that is plotted in the lover levels can be traced to the top level.

Regarding the structure the project management team have chosen to switch the level containing the MCC, GSS and AS with a level containing the 7 different work packages as shown in Figure 6.3. This means that the development of the system is mixed up with requirements to e.g. infrastructure and legal issues. This can endanger the system traceability because the split up into subsystems is not based on a logical split up but a split up based on work packages. The legal issues as an example could have been stated in the analysis forming the objective document and in that way put the necessary constrains on the system level requirements.

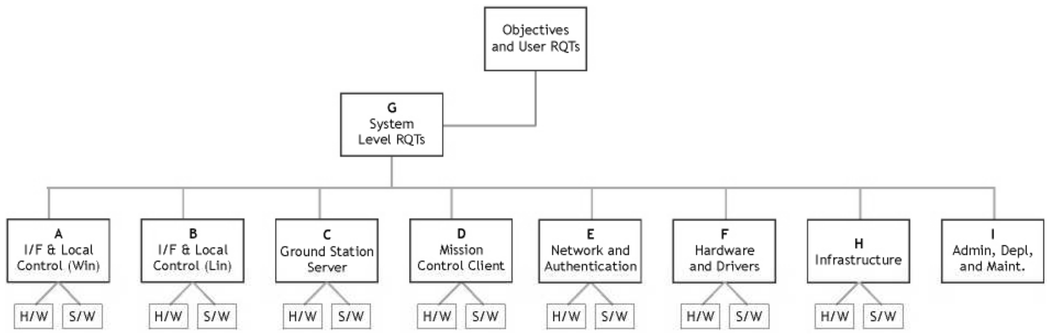


Figure 6.3: The structure of the wiki split into 7 different work packages

The DokuWiki used for the new structure can been seen at http://krevsbech.dk/wiki/doku.php.

6.5 Final system

Because of those decisions and the fact that the group needed to work faster than the GENSO project in the requirement process, it was decided to use the Dokuwiki to create a new structure based on the structure developed for the whole project but only containing the requirements relevant for the MCC. It was chosen to make it from the top and down as the literature prescribes [1]. This implies the following levels in the diversion of the requirements:

- Use Case analysis
- Definition of user requirements
- Definition of system level requirements

It is chosen to split the system level requirements into their regarding subsystem only for the MCC (but also making room for the GSS and AS) and categorize the requirements into modules without making another level. The numbering of each requirement is chosen to afflict structure in Figure 6.4.

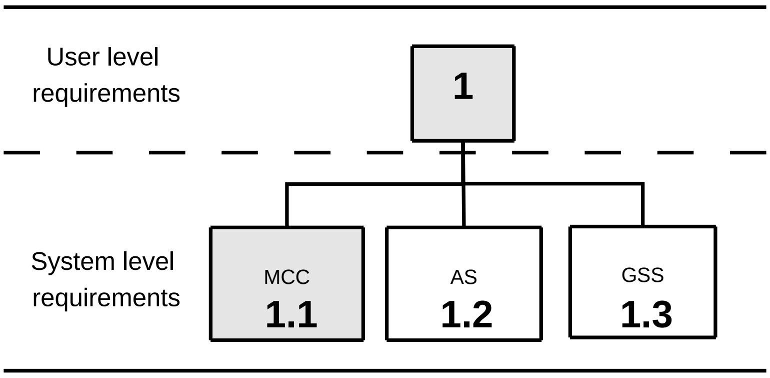


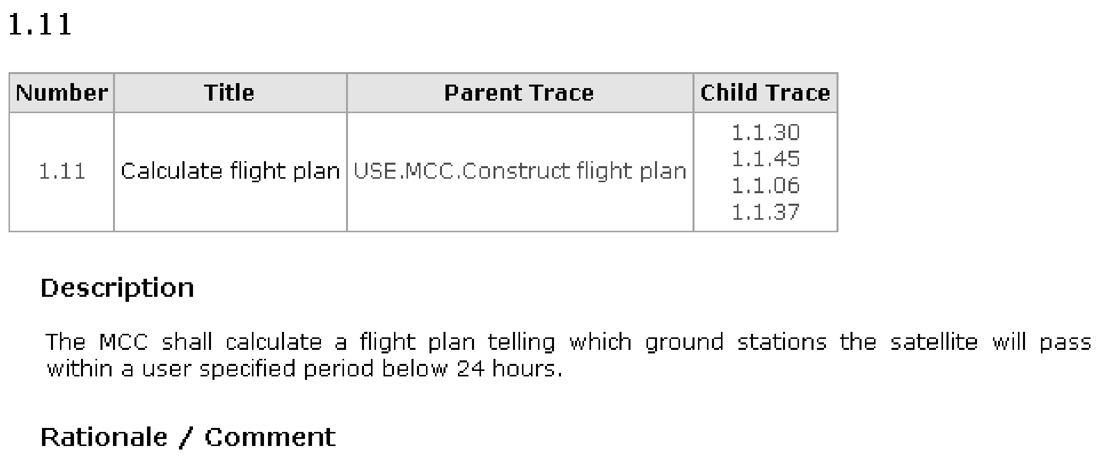
Figure 6.4: The structure of the final system. Only the MCC-requirements of the gray boxes is implemented in the wiki. The numbering of user level is 1.x and system level is 1.1.x

This implies that the user level requirements are numbered as *1.01, 1.02* the system level requirements for the MCC are numbered: *1.1.01, 1.1.02*

The actual definition of the requirements has been done by converting and reformulating the requirements defined by the GENSO team. This is done to be able to make a proper flow down of the requirements. In that process some of the requirements have been discarded due to a wish to implement a base system in the first iteration of the system (See section 5 on page 28)

As described earlier in section 6.2 on page 30, the wiki provide functionality to make a clickable traceability between the different requirements, as shown in Figure 6.5 on the next page this gives a overview of dependencies when something has to be changed. Due the process defining the requirements the project group have discovered that the traceable structure has been valuable. The possibility to trace a system level requirement all the way back to the use case, ease the work when something has to be redefined because it is easier to achieve the overview of which other requirements the change afflicts on.

The new wiki structure can be found at: `http://krav.uninetwork.dk`.



1.11

Number	Title	Parent Trace	Child Trace
1.11	Calculate flight plan	USE.MCC.Construct flight plan	1.1.30 1.1.45 1.1.06 1.1.37

Description

The MCC shall calculate a flight plan telling which ground stations the satellite will pass within a user specified period below 24 hours.

Rationale / Comment

Figure 6.5: An example of a user level requirement in the new wiki structure showing the trace to the use cases (Parent Trace) regarding the requirement and the system level requirements (Child Trace) that is derived from this requirement

It has been tried to implement a description of the interfaces into the Wiki structure, but this work is not entirely finished yet. This means that many of the interfaces has to be derived from the system level requirements, and an Interface Control Document (ICD) must be created to secure that the interface doesn't diverge.

7 Requirement specification

This chapter lists the user requirements to the MCC catagorized after the use cases in chapter 4 on page 23 to make the requirements backwards traceable. There is allocated 10 numbers for the requirements to each use case e.g. from 1.21 to 1.30. This is considered enough to make it possible to add more requirements later on. The accepttest specification describing how to test the requirements is available in appendix C on page 111.

Construct flight plan

1.11 The MCC shall calculate a flight plan telling which ground stations the satellite will pass within a user specified period below 24 hours.

1.12 The user shall be able to see the flight plan as a table containing:

- GSS ID: GENSO administrator defined name
- GSS location: The country and city where the GSS is located.
- Pass start time: UTC time. DD-MM-YYYY HH:MM:SS
- Pass end time: UTC time. DD-MM-YYYY HH:MM:SS

And as a map containing:

- Satellite route printed on the map
- GSS ID: GENSO administrator defined name
- GSS location: Longitude (deg.minute) Latitude (deg.minute)
- Pass start time: UTC time. DD-MM-YYYY HH:MM:SS
- Pass end time: UTC time. DD-MM-YYYY HH:MM:SS

1.13 The user shall be able to chose which GSSes calculated flight plan he attend to use.

1.14 The user shall be able to view the capabilities of each GSS in the flight plan upon request. The capabilities shall be (TBD):

- Frequency band(s)
- Modulation type(s)

Reserve flight plan

1.21 The user shall be able to book the selected GSSes in the flight plan. A booking shall be identified by:

- booking id
- MCC id
- GSS id
- Satellite id
- Start time for the reservation (UTC - unix timestamp)
- End time for the reservation (UTC - unix timestamp)
- Confirmation status of the reservation

1.22 It shall be visible to the user which ground stations have accepted or rejected the reservation.

Communicate with satellite

1.31 During a pass session the MCC shall provide an encrypted uplink channel to the GSS. (TBD)

1.32 During a pass session the MCC shall provide an encrypted downlink channel to the GSS (TBD)

1.33 (Deleted)

1.34 The MCC must store the uplink and downlink satellite mission data in a local database, where a data packet shall be identified by:

- Spacecraft ID (Same as NORAD satellite ID)
- Reception / transmission identifier (flag that defines whether the package is received for the satellite or send to the satellite)
- Reception / transmission node ID (TBC)
- Reception / transmission time: hh:mm:ss dd-mm-yyyy
- Link budget estimation (TBD)
- Doppler-correction factor (TBC)
- S-meter value (if available). (TBC)
- Transmitted data (Defined in the number package transmitted (TBD))

1.35 The GSS shall generate a pass report at the end of a communication session which is requested by the MCC and shown to the user upon request. The report shall be stored locally. The pass report contains:

- Booking id
- Packet count

- Start time of the pass (UTC)
- End time of the pass (UTC)

1.36 The connection status between the MCC and GSS shall be displayed to the user.

1.37 The user shall be able to view how much data has been transferred to and from the satellite in the current session, and in total.

Control GSS

1.41 During a pass session the GSS settings shall be visible to the user. These settings are (TBD):

- Radio frequency
- Doppler correction
- Antenna angle

1.42 Is shall be possible for the user to change the GSS settings

Configure MCC

1.51 The settings of the local MCC shall be visible to the user upon request (TBC)

1.52 The settings and informations regarding the local space craft must be visible to the user upon request These settings are:

- Human-readable spacecraft name
- TLE object number
- Spacelink frequencies, modulations, powers, baud rates, and protocols
- Current MCC online (if there is one)
- Quality estimation of the spacecraft (TBD)
- Whether or not data is currently expected to be transmitted by the spacecraft
- List of affiliated GSSes (TBD)
- Total data downlinked for that spacecraft by the network as a whole
- Total data downlinked for that spacecraft by the affiliated GSS(es) (TBD)

1.53 Is shall be possible for the user to change the settings of the local MCC

1.54 Is shall be possible for the user to change the settings (defined in 1.52) of the local space craft

Logon/logoff network

1.61 The MCC must provide a logon dialog with logon name and password (TBC)

1.62 The MCC must provide a logoff dialog (TBC)

Non use case

1.71 All communication between nodes in the network shall be passed in a human-readable format.

1.72 All communication between software modules shall be passed in a human-readable format.

III Design

8 Design introduction

The overall layout of the GENSO system has already been determined by the GENSO projects management team, as described in 1.1 on page 8. The system consists of MCCs, GSSes and ASs as illustrated on Figure 8.1. Since satellites uses different sets of commands and communication protocols, a satellite operator needs to design his own satellites application (also present on Figure 8.1. This satellite application can then be used for satellite communication once connected to the MCC.

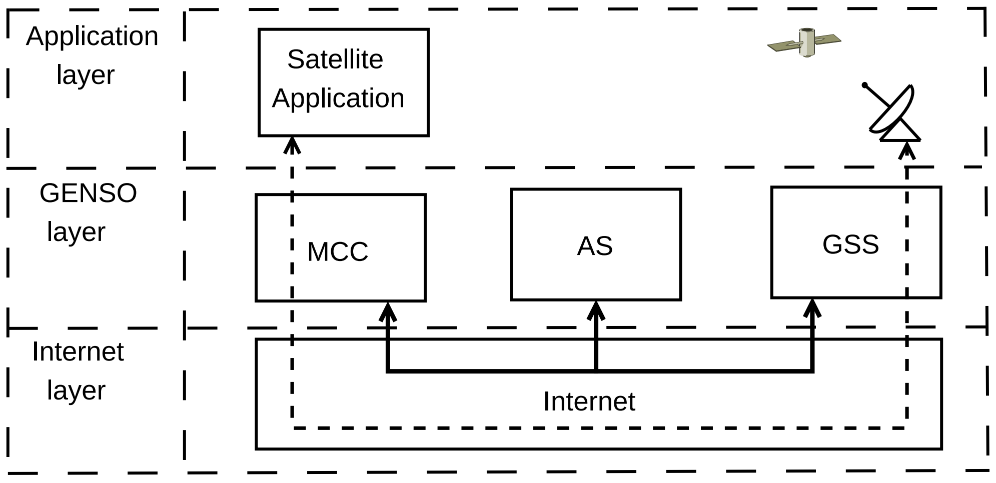


Figure 8.1: The figure illustrates the main idea behind the GENSO project. The figure is divided into 3 logical layers. The application layer, the GENSO layer and the Internet layer. The solid lines shows how the components of the GENSO layer are connected through the Internet. The dotted line shows a data connection sending satellite mission data packets between the satellite application and the satellite.

The application layer consists of the satellite applications and the satellite. The next layer is the GENSO layer consisting of the main components of the GENSO network: MCCs, GSSes and ASs. Beneath the GENSO layer is the Internet layer, which is the media used for communication between the different nodes of the GENSO network.

When a satellite operator wants to communicate with a satellite, the operator uses the MCC to reserve a GSS. Through the MCC the operator informs when the communication with the satellite has to be performed as well as which satellite to communicate with. The MCC analyzes the entered information and calculates where and when the satellite passes a GSS connected to the GENSO network. The result of the calculation is then returned to the satellite operator, and he will know when he'll be able to communicate with the selected satellite. The operator then waits until the satellite passes the selected GSS. When this happens the operator connects the satellite application to a the MCC and the communication with the satellite can begin.

The nodes of the GENSO uses commands (or messages) to communicate with each other. These commands are sent through the Internet as illustrated on Figure 8.2.

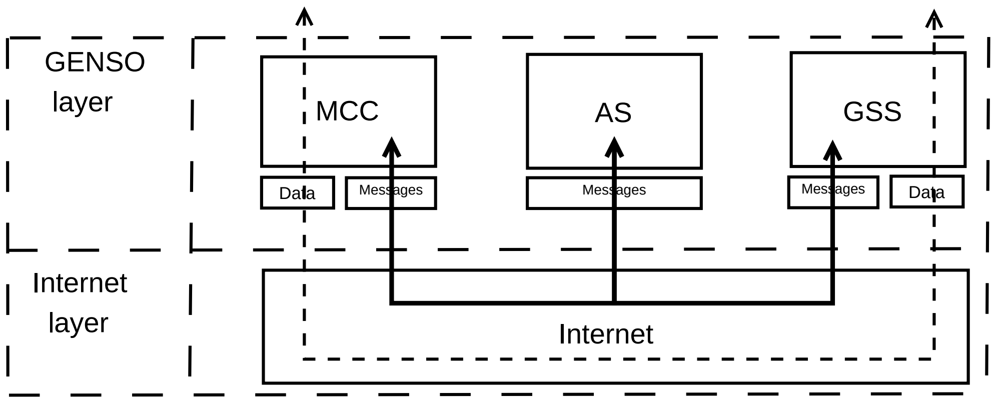


Figure 8.2: The figure illustrates the components of the GENSO network. The solid lines represent the messages sent between the nodes. The dotted lines represent satellite mission data packets sent between the satellite application and a satellite. As shown the messages and the satellite mission data packets are received on different channels and treated differently. The satellite data packages is forwarded to the above layer.(see Figure 8.1) The messages is handled at the "GENSO layer"

Each type of node on the GENSO network has a set of tasks it has to fulfill. The MCC is the satellite operators access point to the GENSO network. By using the MCC the satellite operator can send messages to other nodes of the GENSO network. The AS is responsible for authenticating the nodes on the network. Storing list of participating GSSes and satellites is also managed by the AS. That way a MCC can look up the address of a GSS when needed. Each MCC and GSS has to be authenticated by the AS before they are registered as active GENSO nodes. Once a node is authenticated, inter node communication from and to that particular node, is possible.

The GSSes are the nodes responsible for the communication with a satellite. The satellite operator uses the MCC to send satellite mission data to a GSS and these data are then sent to the satellite by the GSS.

As described in the problem formulation (see chapter 3 on page 21) this project is concerned with the MCC. The rest of this chapter explains the design of the MCC.

Before describing each individual module in further details, some general design methods used in the design of the MCC. These method has influence on each module, which is why these methods are introduced first.

9 Design methods

The GENSO network is designed as a distributed system of nodes communicating with each other. To explain this, and to enable the possibility of executing some of the MCC's modules on separate hardware platforms, this is the main focus of section 9.1 where the distributed design approach is explained in further details.

The inter modular messages between modules of the MCC shall use a human-readable format as stated by user level requirement 1.72. It has been chosen to include this in the design by the use of XML. Section 9.2 explains how XML is included in the design of the MCC.

9.1 Distributed system

As specified previously the GENSO network consists of several types of nodes, namely an AS, several MCCs and several GSSes. This implies that the GENSO network is a distributed system using the Internet as the communication backbone.

Each node that can be contacted by other nodes in the network (in GENSO only AS and GSS) has to specify it's global IP-address and port. Once a transmitting node knows the IP and port of a receiving node, messages can be exchanged between the two nodes. The information about other nodes' contact information, must be retrieved from a always known location, and this is one of the main purposes of the AS.

The AS keeps track of each nodes contact information. Since it is only the GSSes that are contacted, not the MCCs, the contact informations are kept in the GSSL. For each GSS the GSSL contains:

- IP-address
- TCP/IP port
- GSS ID

as well as several additional values for each node (the GSSL is defined by the GSS and is therefore To Be Determined (TBD)). This way each node only needs the IP-address and port of the AS to know how to connect to all additional nodes on the network. Since all nodes needs the GSSL it's practical to store this list centrally at the AS.

Another type of information that several nodes (particularly GSS's) on the networks needs, is information of each satellite associated with the GENSO network. When a certain GSS needs to communicate with a satellite, the GSS has to know which frequency and modulation to use for the communication. The AS stores these values for each satellite in the PSL. That way a GSS only needs to ask the AS to transmit the values for given satellite, before the GSS communication is possible.

The AS in this distributed system has the potential risk of being a “single point of failure”, and this needs to be considered. A MCC in the GENSO network would not be able to contact any GSSes without the GSSL, but if the GSSL has been received once, only GSSes that have changed their contact info will be unreachable. Authentication is another issue, and although this is not considered in this project, it will be a problem if the AS is unavailable.

Although the GENSO network has a single point of failure risk, the distributed design solves problems discovered in the GSMS system (see section 2.3.2 on page 19). The centralized scheduling calculation in GSMS is distributed in GENSO because it is accomplished as a cooperation between the GSSes and one MCC at a time instead.

In the chosen design it is not only the network that is distributed, this method is also used in the module design of the MCC.

As stated in user level requirement 1.72 (see section 7 on page 37), the software modules must be designed to communicate in a human readable format. Implementing such communication between independent software modules, must be done with some kind of Inter Process Communication (IPC). If this IPC is chosen to be socket communication, the modules can actually be distributed across a network as well as the nodes of the GENSO network. Since the MCC will be used in various types of setups, this could be useful. Then it would be possible to have some of the MCC’s tasks executed on separate hardware platforms.

As an example the MCC uses a database for data storage. This database might be located on a separate hardware platform, and the rest of the MCC modules gathered on another hardware platform. This provides a MCC with a flexible design, allowing the MCC maintainer to choose freely how to run it.

Separating the modules implies that each module needs to know how to contact the other modules. This can be solved in with different approaches, but in this project, and central message handler is chosen to facilitate the communication. That way each module only has to know the message handler, and the message handler can acts as a message router between the modules, based upon module ID.

Now that the distributed design is explained, a way to solve the communication in a human readable format must be chosen.

9.2 Using XML for communication

It has been chosen to use a particular way of communicating between the nodes in the network (MCCs, GSSes and ASes) as well as between the different software modules in the MCC. This originates from the user level requirement 1.72 (see section 7 on page 37) stating that nodes and modules must communicate in a "human readable" format. It has been chosen to use XML to fulfil this requirement. This has already been proved succesfully used in the Mercury Ground Station Network (MGSN) (see section 2.3.1 on page 18)

XML is explained like this at www.wikipedia.org (quote [16]):

> *The Extensible Markup Language (XML) is a W3C-recommended general-purpose markup language that supports a wide variety of applications. XML languages or 'dialects' are easy to design and to process. They are also reasonably human-legible, and to this end, terseness was not considered essential in its structure. XML is a simplified subset of Standard Generalized Markup Language (SGML). Its primary purpose is to facilitate the sharing of data across different information systems, particularly systems connected via the Internet.*

XML is defined by W3C[1] in RFC3023 [2]

The reason for using XML is explained in the following sections

9.2.1 Features of XML

XML is a way of using text to describe and apply a tree-based structure to information. This means that it is possible to save information as text, which is split into a hierarchy of "elements" and "attributes" for these elements. The structure is achieved by the means of so called markup, to indicate where a certain type in the document starts and stops. An example is given here:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
  <title>Basic bread</title>
  <ingredient amount="3" unit="cups">Flour</ingredient>
  <ingredient amount="0.25" unit="ounce">Yeast</ingredient>
  <ingredient amount="1.5" unit="cups" state="warm">Water</ingredient>
  <ingredient amount="1" unit="teaspoon">Salt</ingredient>
  <instructions>
    <step>Mix all ingredients together, and knead thoroughly.</step>
    <step>Cover with a cloth, and leave for one hour in warm room.</step>
    <step>Knead again, place in a tin, and then bake in the oven.</step>
  </instructions>
</recipe>
```

[1] http://www.w3.org/XML/
[2] http://www.ietf.org/rfc/rfc3023.txt

[16]

The first line specifies the XML document's "declaration". This is optional, but gives information about what version of XML is used, and the character encoding. The rest of the document contains nested elements in the form:

```
<name attribute="value">content</name>
```

This means that each element has a start tag and an end tag, possibly surrounding other elements or content. The start tags consists of a name surrounded by angle brackets (e.g. `<step>`) and can also include attributes (e.g. `<step attribute="step1">`) which is specified like name-value pairs. The attribute values must always be quoted with either single or double quotes. The end tag consists of the same name surrounded by angle brackets but with a leading "/" (e.g. `</step>`). The content of an element can be everything that appears between the start and end tag, including other "child" elements or text.[16]

An advantage when using XML to communicate, is that it is programming language independent. Any language can implement the needed parsers and lexers to be able to read and manipulate the XML. When the XML then is formed from a schema file, both sides of the communication agrees upon the format, and can exchange the needed information.

Actually it is not really needed to use a programming language, because one could just sent the right formed XML as text from an application e.g. Telnet[3].

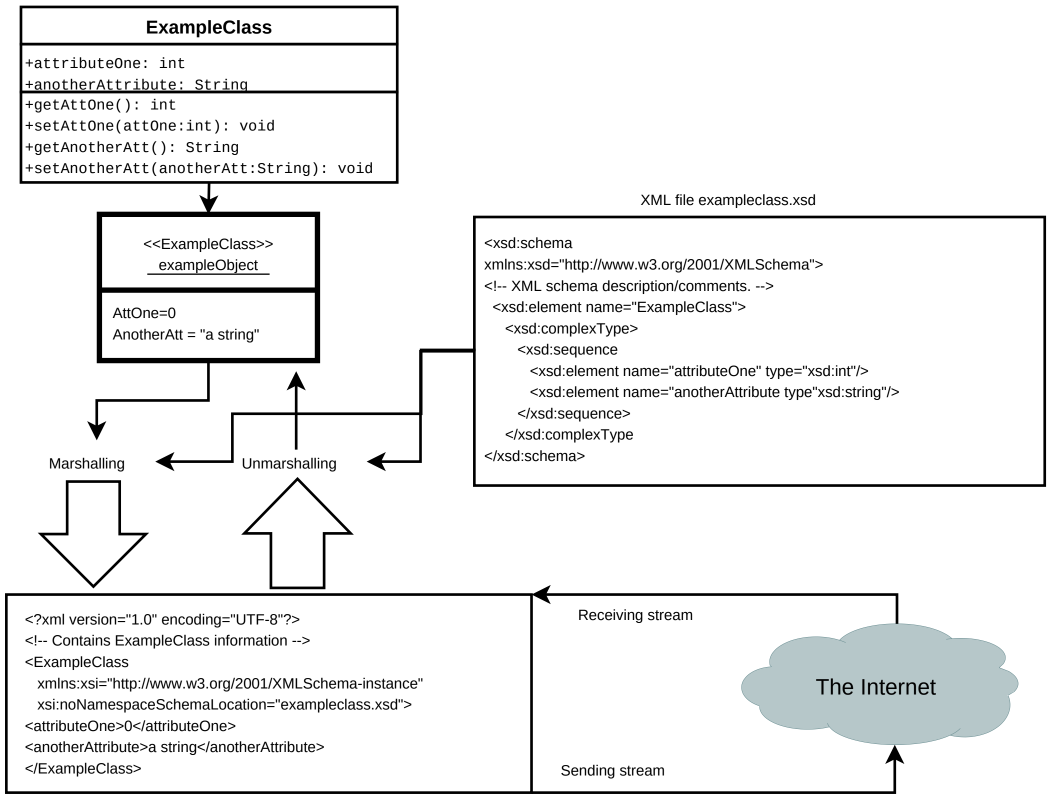
Another advantage is that the XML is in a so called "human readable" format. This means that the messages exchanged can be interpreted by humans without translations. This can make debugging and testing easier.

9.2.2 XML and Object Oriented Programming

Since it is chosen to implement the Mission Control Client (MCC) in an object oriented programming language, it is useful to know how XML can be used in an object oriented way. Each different object oriented programming language implements the usage of XML through their own APIs, but the XML that is parsed is the same.

The figure below shows how XML can be used in an object oriented way:

[3]http://www.ietf.org/rfc/rfc0854.txt?number=854



```
+attributeOne: int
+anotherAttribute: String
+getAttOne(): int
+setAttOne(attOne:int): void
+getAnotherAtt(): String
+setAnotherAtt(anotherAtt:String): void
```

```
<xsd:schema
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!-- XML schema description/comments. -->
  <xsd:element name="ExampleClass">
    <xsd:complexType>
      <xsd:sequence
        <xsd:element name="attributeOne" type="xsd:int"/>
        <xsd:element name="anotherAttribute type"xsd:string"/>
      </xsd:sequence>
    </xsd:complexType
</xsd:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Contains ExampleClass information -->
<ExampleClass
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exampleclass.xsd">
<attributeOne>0</attributeOne>
<anotherAttribute>a string</anotherAttribute>
</ExampleClass>
```

Figure 9.1: The figure shows an abstract model of how to use XML for data and communication exchange in an object oriented way. The xml schema file (called exampleclass.xsd) is used to translate the object ExampleClass into XML that can be transmitted over the internet as a stream. The process of translating an object to a xml file, is called to "marshall" the object, and the other way is called to "unmarshall" it. The class in the example is java specific.

Normally one would construct an object to hold the complex information that needs to be stored and/or sent to make the application functional. This is also possible when using XML, since the object is translated into XML directly using a schema (sometimes called "binding definition"). So once the schema file for the translation is constructed the class with the needed info can be auto generated by the programming API. With an instance of the auto generated class, the data can be saved as an object, and then the process of "marshalling" the data in the object to XML is a matter of calling the programming API marshalling routine. Then the XML can be sent via network, as a file, or as it is usually done, via the Internet. The receiving side can then read the XML (file, stream etc.) and call the "unmarshalling" routine. Then the receiving side has the same object to manipulate as started out in the sending side.

9.2.3 Summary

XML fulfil the given user level requirement by being human readable. At the same time XML is a standard communication language, already implemented in various programming language with usable APIs. It is a structured language, that makes it possible to represent various datastructures, and it is possible to use XML in an object oriented way. This makes XML usable for message communication in the GENSO project.

The following chapter will introduce the module design of the MCC, using XML for communication between modules.

10 Module design

This chapter describes the design of the individual modules of the MCC. Based on the classification of the requirement specification in chapter 7 on page 37, the MCC has a logical structure of internal software modules. The requirement specification includes requirements about security, i.e. encrypted data channels and network logon. It has not yet been specified which encryption algorithm to use or how to authenticate nodes in generel in the GENSO network, thus this project will not try to solve any security issues.

The modules of the MCC is shown in Figure 10.1 which is a view of the MCC as a clip from Figure 8.2 on page 43 with communication arrows to the network and between modules. This figure will be used throughout the module design in the description of each module with a highlighting of the current module and arrows to the interfacing modules.

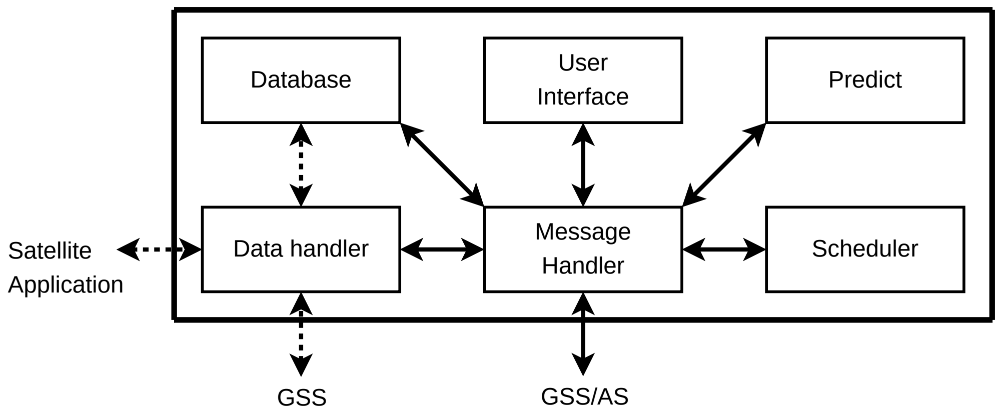


Figure 10.1: The modules of the MCC. The dotted arrows are binary satellite mission data and the solid arrows are XML messages. The figure will be used throughout the module design to highlight the current module.

Before the design sections of each module, a short description of the modules is given to give an overview of the functionality. The MCC modules handles the following tasks:

User Interface
The User Interface module handles the interaction between the user (operator) and the GENSO network. The User Interface module is implemented as a "thin client", where all functionality is actually carried out by the underlying modules. This means that a "book pass" button will send an XML message to the Scheduler module. Implementing it like this, gives the possibility to choose any other tool to send these messages. The User Interface module is described in detail in section 10.1 on page 54.

Message Handler
The Message Handler module provides the interface for internal module communication and internode communication in the GENSO network. The Message Handler acts as a "message router" by forwarding messages from internal modules to GSS and AS nodes in the network, and vice versa. The Message Handler also provides a "debug logon" facility to view all messages sent in the MCC. The Message Handler module is described in detail in section 10.2 on page 65.

Scheduler
The Scheduler module handles the scheduling of GSS bookings. It provides the functionality to use the Predict module to get information about available passes, and then present this to the user via the User Interface module. The user can then choose which passes to book, and the Scheduler module takes care of reserving these passes at each GSS. The Scheduler module is described in detail in section 10.3 on page 72.

Predict
The Predict module handles the prediction of satellite passes over ground stations in the network. This is achieved by communicating with the predict library, which makes the actual prediction calculation, and then handing the calculated information to the Scheduler module. The Predict module is described in detail in section 10.4 on page 84.

Database
The Database module provides an interface for storage of satellite mission data as well as booking information and pass reports. The Database module is described in detail in section 10.5 on page 89.

Data Handler
The Data Handler provides a link between the GSS and the satellite application of the user. It handles all the satellite mission data and forwards uplink from the satellite application to the GSS and the Database module, and forwards downlink from the GSS to the Database module and the satellite application. The Data Handler module is described in detail in section 10.6 on page 96.

UML view

UML offers two basic levels for encapsulation of functionality in object oriented software development: Classes and packages. A class is an essential part of modeling the behavior of a system. It describes an object type with regard to properties and behavior.[5] This way it is possible to construct a model of real world elements which can be a high level abstraction of source code structure. It makes it easier for non developers, but also developers to understand the system for presentational purposes. The syntax of attributes and methods in the class diagrams in this report follows the syntax of Java expressions as Java is chosen for implementation language in this project.

A package in UML is a collection of other UML elements like classes or even other packages. It also provides a namespace for the grouped elements which makes it possible to have more elements with the same name, but in different packages.[5]

In this project the modules of Figure 10.1 on page 50 are regarded as packages and each package will be a grouping of classes. Figure 10.2 shows a diagram of the packages in the system and the relationship between them. The distributed design choice with a possibility to split the modules into different locations implies that every module must be an individual program or process.

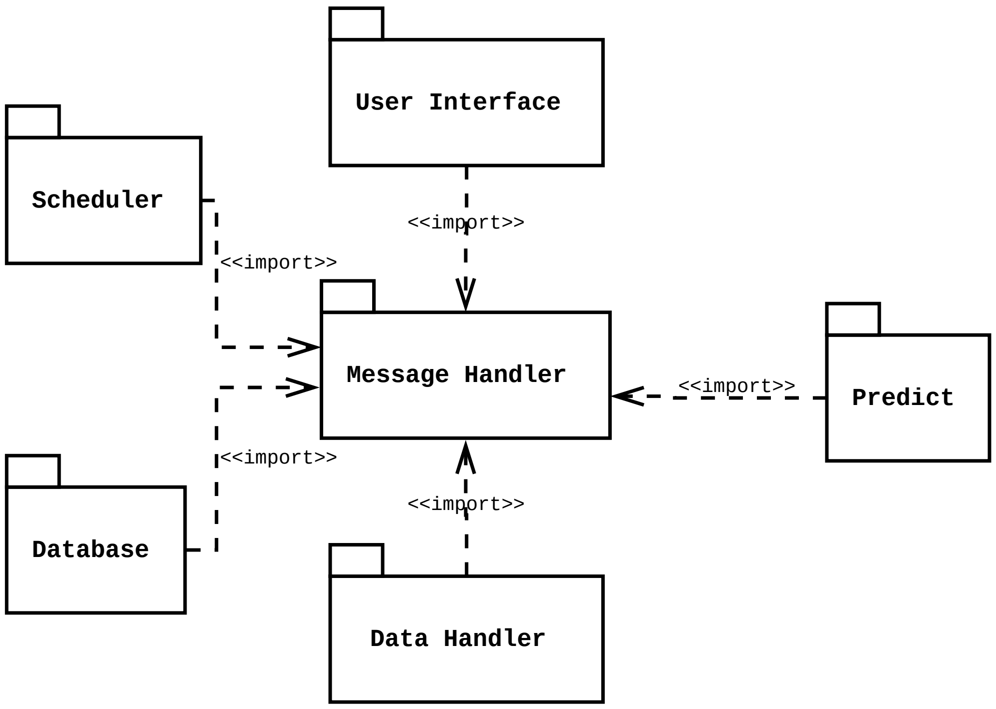


Figure 10.2: The packages/modules in the MCC. Every package in the system depends on the package "Message Handler" to communicate with each other and needs to import communication methods from it.

Module design recipe

The design of each module will be defined in a generic recipe with the following subsections:

Module figure: A module figure like Figure 10.1 on page 50 with highlighting and interface arrows.

Requirements: This will list and discuss the system level requirements of the current module to specify in detail what the module shall do.

Descriptions and diagrams: This will discribe the flow(s) of the module in UML activity diagrams and the overall design of the module as a UML class diagram.

Interfaces: This will list and discribe the interfaces to the module related to input and provided services.

Test specification: This will specify a scenario for testing of the module functionality.

In the following sections the modules of the MCC is described

10.1 User Interface

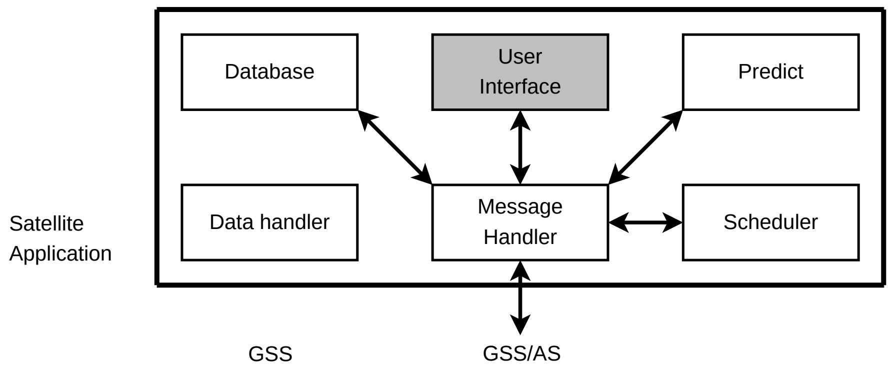


Figure 10.3: The figure shows the User Interface modules in relation to the rest of the MCC. User Interface has interfaces to all modules except the Data Handler module.

Figure 10.3 shows the User Interface module in the MCC. To be able to make use of the GENSO network there has to be an interface available for the user. This is accomplished by the User Interface module. The overall functionality of the User Interface module is listed in the following:

- Configure the MCC
- Show satellite information
- Show the GSSes that can be used for communication with a specified satellite
- Reserve the wanted GSS
- Adjust the radio and antenna configuration at the GSS

To be able to provide this functionality, the User Interface module makes use of several other modules: The Scheduler module, the Message Handler module, the Database module and the Predict module. The User Interface module also communicates with AS and GSS, to accomplish some of its tasks.

Besides the overall functionalities, it is the User Interface modules task to make sure that the user input is valid, so possible failures are avoided throughout the system.

10.1.1 Requirements

The requirements stated in the user level requirements in chapter 7 on page 37 are derived to nine system level requirements, that define the functionality of the User Interface module. User level requirements applicable for the User Interface module are; 1.11, 1.12, 1.21, 1.34, 1.35, 1.41, 1.42, 1.51, 1.52, 1.53.

1.1.01 The User Interface module shall present an overview of previous satellite sessions that are stored in the locale database. A satellite session is the data that has been received from a satellite, and must be identified by the following:

Item	Data type:
Satellite ID	String
GSS ID	String (TBD)
Booking information ID	String (TBD)
Pass Start time	UNIX time stamp (TBD)
Pass End time	UNIX time stamp (TBD)

To be able to have an overview over previously executed sessions, it is necessary for the user to be able to distinguish between them. Therefore the User Interface module needs to be able to present the above information for the user. This requirement descents from user level requirements 1.34 and 1.35.

1.1.02 The User Interface module shall present a list of GSSes that the satellite passes within a given time period. This information must be presented to the user in two formats:

- A table
- A two dimensional map

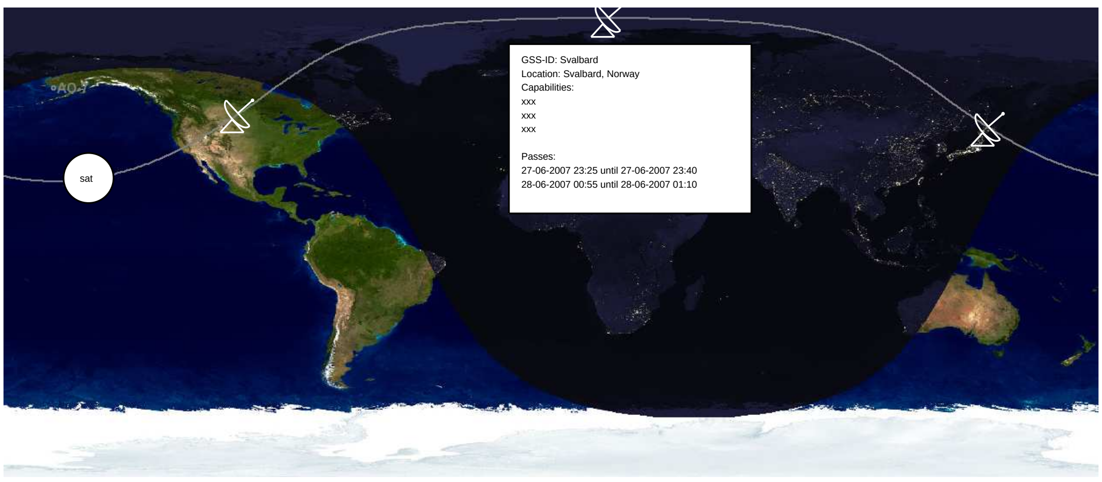
The table is created based on the following:

Item	Data type:
GSS ID	String (TBD)
GSS location	String (TBD)
Pass start time	UNIX time stamp (TBD)
Pass end time	UNIX time stamp (TBD)
Frequency band	Integer array (TBD)
Modulation type	String array (TBD)
Capabilities	String (TBD)
(TBD)	(TBD)

The two dimensional map is created from the following information:

Item	Data type:
GSS location	String (TBD)
Pass start time	UNIX time stamp (TBD)
Pass end time	UNIX time stamp (TBD)
Frequency band	Integer array (TBD)
Modulation type	String array (TBD)
Capabilities	String (TBD)
GSS location coordinates	Float array (TBD)

A raw sketch of the map view is given in Figure 10.4:



GSS-ID: Svalbard
Location: Svalbard, Norway
Capabilities:
xxx
xxx
xxx

Passes:
27-06-2007 23:25 until 27-06-2007 23:40
28-06-2007 00:55 until 28-06-2007 01:10

Figure 10.4: The figure shows a raw sketch of how the map presentation should be implemented. The satellite path with available passes over three different ground stations can be seen, as well as how it is possible for the satellite operator to view details of the ground stations.

This requirement is derived directly from the user level requirement 1.12.

1.1.03 The user shall be able to reserve a GSS, based on the following parameters:

Item	Data type:
Satellite ID	String (TBD)
GSS ID	String (TBD)
MCC ID	String (TBD)
Booking ID	Integer
Start time for reservation	UNIX time stamp
End time for reservation	UNIX time stamp
Confirmation status of the reservation	Boolean
Received by GSS status	Boolean

These parameters combined with information about which GSS(es) the user has selected, are used to create a booking information. A booking information contains all the necessary information to create a booking (TBD). This requirement is derived from user level requirement 1.21.

1.1.04 The user must be able to configure the radio and antenna at the GSS he is currently connected to, through the local User Interface module. The configurable parameters are:

Item	Data type:
Radio frequency	Float (TBD)
Doppler correction	Float (TBD)
Antenna angle	Float (TBD)

1.1.05 The User Interface module must show the current GSS configuration. The configuration parameters are defined in system level requirements 1.1.04.

To be able to make necessary calibrations while a session is active, the user is given the option to make manual adjustments to the GSS' communication hardware. User level requirement 1.41 and 1.42 derived these two requirements. As it is not clarified which functionality the GSS offers, all available parameters are not represented.

1.1.06 The User Interface module must provide fields for the user to enter the flight plan options. These options must be contained in the following form:

Item	Data type:
Satellite ID	String (TBD)
Flight plan start time	UNIX time stamp (TBD)
Flight plan end time	UNIX time stamp (TBD)

The Scheduler module needs the time window and satellite ID where the user wishes to contact the satellite, to be able to locate the GSSes that the user establish a communication session with.

1.1.07 The User Interface module shall retrieve the current MCC configuration and display it to the user. The content of this configuration is defined in user level requirement 1.51.

1.1.08 The User Interface module must be able to configure the MCC. These configuration parameters are given in 1.51.

These two system level requirements are derived from user level requirement 1.51 and 1.53, respectively. The configuration parameters are all marked with TBD, so no specification can be given, before it is clarified what can be configured on the MCC.

1.1.09 The User Interface module must present the configuration for the selected satellite. The contents of this configuration is defined in 1.52.

This requirement is derived from user level requirement 1.52. The PSL contains information about a given satellite, static as well as dynamic in form of statistics. This requirements gives the user an overview of a given satellite.

10.1.2 Descriptions and diagrams

The main functionalities of the User Interface module are illustrated in this section through activity diagrams.

Initialization and creation of the User Interface module

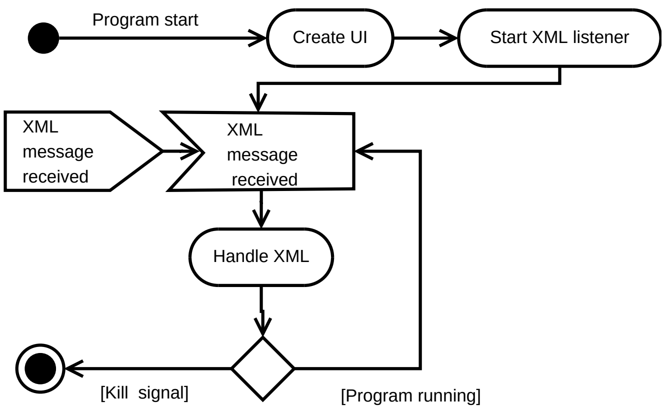


Figure 10.5: Activity diagram of the initialization of the User Interface module

Figure 10.5 shows how the User Interface module is initialized, when the user starts the module. The flow is as follows:

1. The user interface is created
2. An XML listener is started
3. Incoming XML messages are handled, until a kill command interrupt is received.

Request and show available GSSes

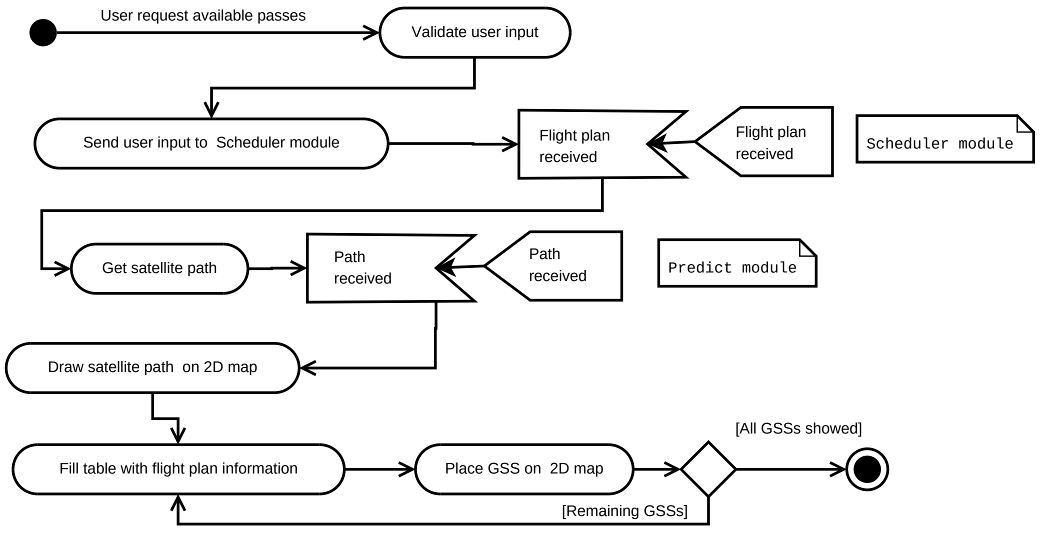


Figure 10.6: Activity diagram of how available GSS are showed

When the user requests satellite passes for a given satellite within a given time span the activity diagram in Figure 10.6 is executed. The flow is as follows:

1. The input is validated to see whether the user has entered a valid time period.
2. The request is sent to the Scheduler module for calculation of available passes.
3. The User Interface module requests a satellite path for the given satellite within the given time period from the Predict module.
4. When both the path and available GSSes have been received, the User Interface presents it all to the user.

Reserve GSSes

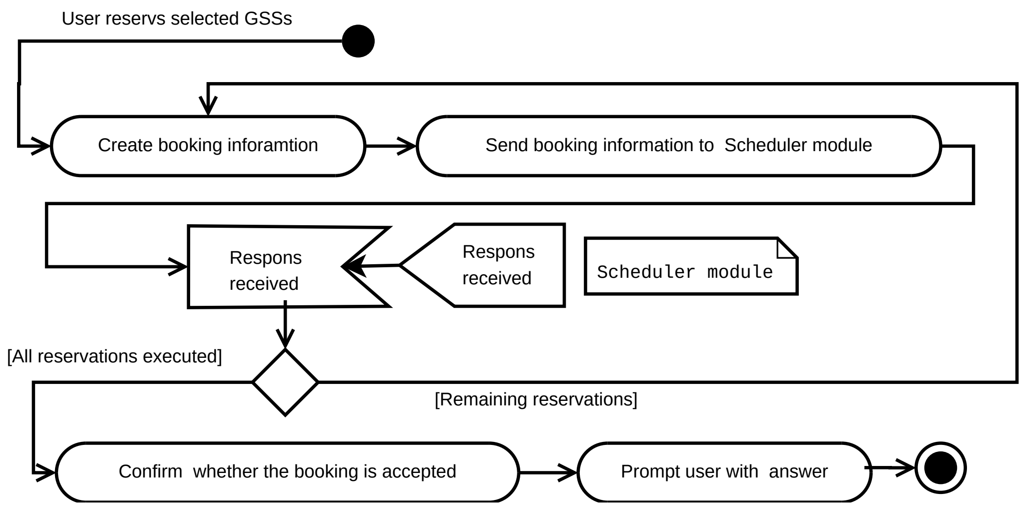

User reservs selected GSSs
Create booking inforamtion
Send booking information to Scheduler module
Respons received
Respons received
Scheduler module
[All reservations executed]
[Remaining reservations]
Confirm whether the booking is accepted
Prompt user with answer

Figure 10.7: Activity diagram of how a reservation is created

After the presentation of available GSSes, the user can select and reserve the ones that are relevant. The reservation procedure can be seen in Figure 10.7. The flow is as follows:

1. For each GSS the user has selected, a booking information is created.
2. The booking information is sent to the Scheduler module.
3. When all booking replies have been received, the information is showed to the user.

Show old sessions

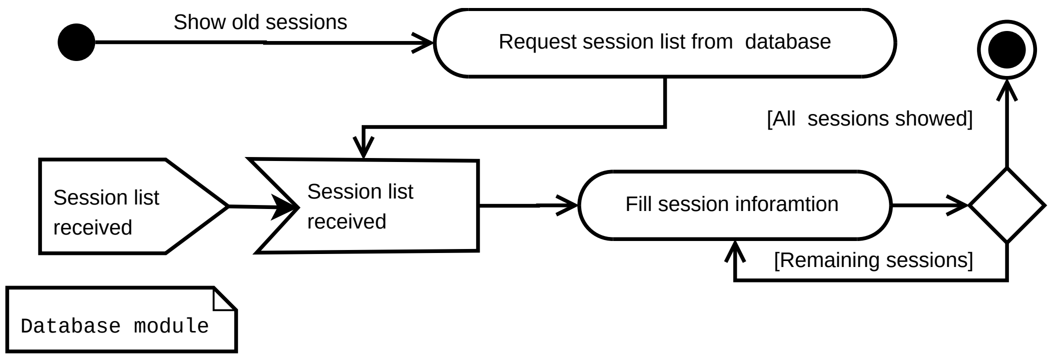


Figure 10.8: Activity diagram of how old sessions are retrieved

Figure 10.8 shows the flow of viewing old sessions that are available in the database. The flow is:

1. The User Interface module requests old sessions from the Database module.
2. When the sessions are received they are all showed to the user.

The system level requirements and the activities above can be classified into five different categories that together cover the functionality of the User Interface module. Each main category is defined in the table below:

Category:	system level requirements:
Configure the MCC	1.1.07 and 1.1.08
Show satellite information	1.1.09
Show and reserve GSS	1.1.02, 1.1.03 and 1.1.06
Configure the MCC	1.1.04 and 1.1.05
Show overview of old sessions	1.1.01

These categories derive the five class diagrams that can be seen in Figure 10.9 on the next page.

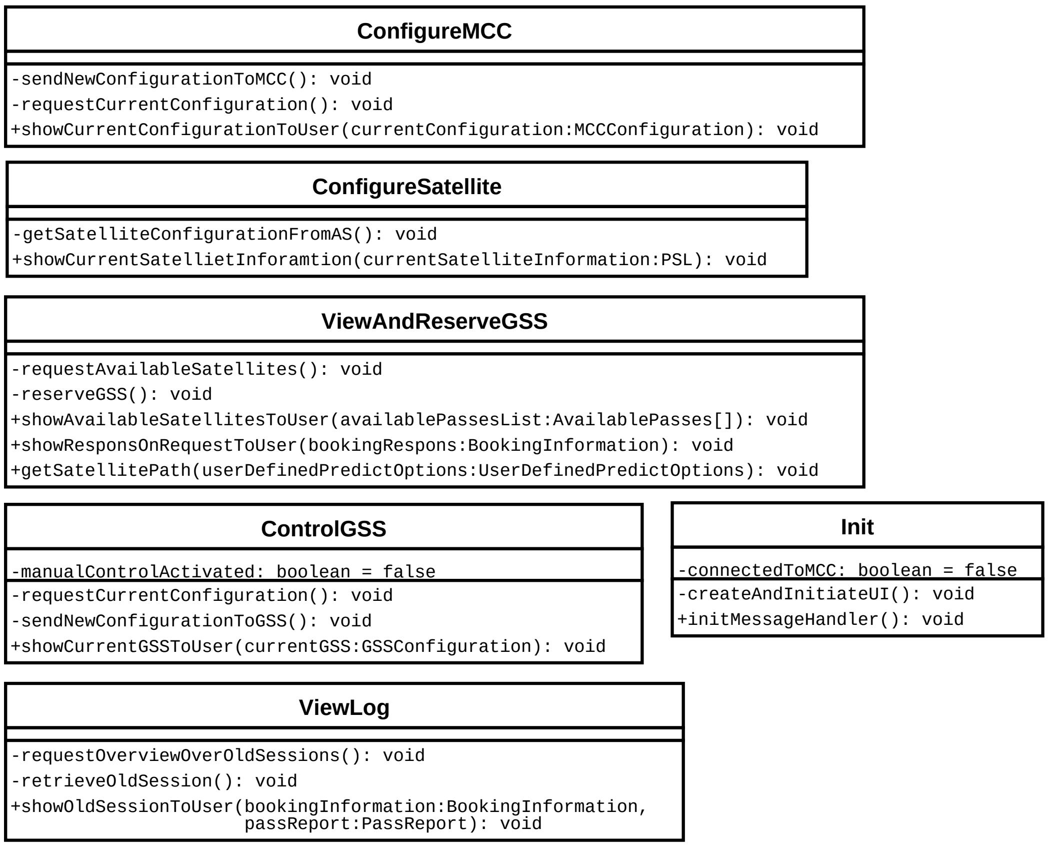

ConfigureMCC
-sendNewConfigurationToMCC(): void
-requestCurrentConfiguration(): void
+showCurrentConfigurationToUser(currentConfiguration:MCCConfiguration): void
ConfigureSatellite
-getSatelliteConfigurationFromAS(): void
+showCurrentSatellietInforamtion(currentSatelliteInformation:PSL): void
ViewAndReserveGSS
-requestAvailableSatellites(): void
-reserveGSS(): void
+showAvailableSatellitesToUser(availablePassesList:AvailablePasses[]): void
+showResponsOnRequestToUser(bookingRespons:BookingInformation): void
+getSatellitePath(userDefinedPredictOptions:UserDefinedPredictOptions): void
ControlGSS
-manualControlActivated: boolean = false
-requestCurrentConfiguration(): void
-sendNewConfigurationToGSS(): void
+showCurrentGSSToUser(currentGSS:GSSConfiguration): void
Init
-connectedToMCC: boolean = false
-createAndInitiateUI(): void
+initMessageHandler(): void
ViewLog
-requestOverviewOverOldSessions(): void
-retrieveOldSession(): void
+showOldSessionToUser(bookingInformation:BookingInformation, passReport:PassReport): void

Figure 10.9: Class overview for the User Interface module package

Beside the five classes that illustrate the above mentioned categories, there is a `init` class that takes care of creating the user interface and initializing a XML message handler. This is needed to receive responses on the request made by the user. Depending on what the user wants to do, a corresponding class is utilized to execute the needed functionality.

10.1.3 Interface

The interface section covers which modules the User Interface module needs to interact with, to provide the above mentioned functionality. The User Interface module's interface within the MCC are described first.

The Scheduler module

When the list containing possible satellites has to be shown, the Scheduler module sends a flight plan. The flight plan must contain the following:

```
<FlightPlan>
```

```
    <gssID></gssID>
    <gssLocation></gssLocation>
    <startTime></startTime>
    <endTime></endTime>
    <gssFrequencyBand></gssFrequencyBand>
    <gssModulationType></gssModulationType>
    <gssCapabilities></gssCapabilities>
    <gssLocationLatitude></gssLocationLatitude>
    <gssLocationLongtitude></gssLocationLongtitude>
    <gssLocationAltitude></gssLocationAltitude>
  </FlightPlan>
```

In addition, the Scheduler module is able to send a response on GSS reservations. This must look like the following:

```
<bookingList>
 ...
  <BookingInformation>
    <bookingID></bookingID>
    <gssID></gssID>
    <mccID></mccID>
    <satID></satID>
    <startTime></startTime>
    <endTime></endTime>
    <receivedByGSS></receivedByGSS>
    <timeslotConfirmed></timeslotConfirmed>
  </BookingInformation>
 ...
</bookingList>
```

Database module

When the Database module sends an overview of old sessions, it sends two lists, a booking information and a pass report. The booking information can be found just above, while the pass report is defined in section 10.5.3 on page 94.

To be able to collect some of the information that needs to be showed, the User Interface module needs to interact with instances of the GSS and AS:

AS

The AS can send a PSL, to show the information about the different satellites.

```
  <PSL>
    <satelliteID></satelliteID>
    <tleObjectNumber></tleObjectNumber>
    <spacecraftInformation>
        <frequency></frequency>
        <modulation></modulation>
        <power></power>
        <baudRate></baudRate>
        <protocol></protocol>
    </spacecraftInformation>
    <currentMCCOnline></currentMCCOnline>
    <qualityEstimation></qualityEstimation>
    <listOfAffiliatedGSS>
        ...
    </listOfAffiliatedGSS>
    <totalDownlinkData></totalDownlinkData>
    <totalUplinkData></totalUplinkData>
</PSL>
```

GSS

To make it possible to give the user an overview of the current radio and frequency settings the GSS has to send the current configuration to the User Interface module. The following gives a suggestion to the XML:

```
<GSSConfiguration>
    <frequency></frequency>
    <dopplerCorrection</dopplerCorrection>
    <antennaAngle></antennaAngle>
</GSSConfiguration>
```

10.2 Message Handler

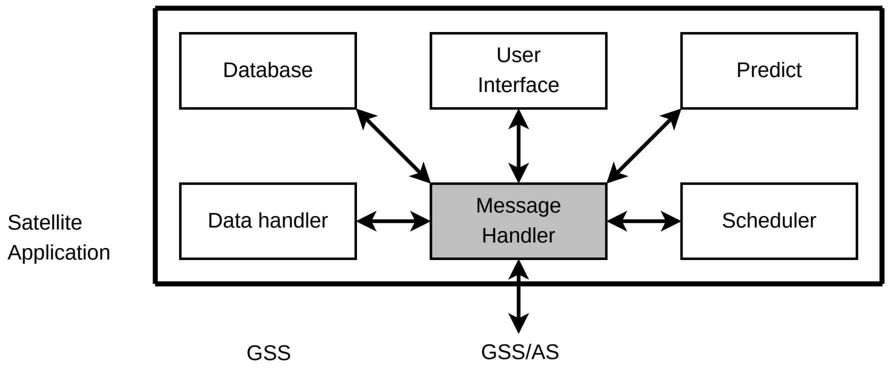


Figure 10.10: The modules of the MCC. Current module is the Message Handler that interfaces to all other modules in the MCC and to the GSS and AS

Figure 10.10 shows the Message Handler module in the MCC. The Message Handler module is responsible for controlling inter node as well as inter module communication. The point of having a Message Handler module is that other nodes in the GENSO network only have to communicate with one module, and having a central module to handle module communication instead of creating a direct connection from module to module. This means that the Message Handler module will receive XML messages from instances of the GSS and from instances of the AS. These messages will be forwarded to the appropriate modules. The Message Handler module will also be responsible for sending XML messages to other nodes in the network, meaning that another module in the MCC sends a command to the Message Handler module, which forwards it to the appropriate instance of the GSS or AS. The Message Handler module will act as a “message router” in the MCC, meaning that all messages will go through here. This gives a possible debugging facility, if it is possible to log on to the Message Handler module and view all messages sent.

10.2.1 Requirements

User level requirements 1.71 and 1.72 has requirements to the Message Handler module (see 7 on page 37). A set of system level requirements has been constructed to match the user level requirements. The system level requirements are explained one by one with reference to the user level requirements.

1.1.15 When a GENSO XML message is sent to another node in the network the Message Handler module shall keep the connection and wait for a reply until this is received as GENSO XML, or until 10 seconds (TBD) timeout period. The message will be a stream of characters.

1.1.16 The Message Handler module shall, upon request from other modules, send all outgoing

GENSO XML messages. These messages shall be sent over a TCP/IP connection on the port specified in the GSSL or ASL to the IP address given in either the GSSL or the ASL, as a stream of characters.

These requirements relates to user level requirement 1.71 stating that the nodes in the network have to communicate in a human readable format. To achieve this a format of the readable text has to be chosen. This format is defined in the interface requirement (i.6) regarding GENSO XML:

i.6 The GENSO XML shall be defined as a .xsd file, derived from the w3 standard for XML schemes:

http://www.w3.org/2001/XMLSchema

The XML version shall be 1.0 and the encoding of characters shall be UTF-8

The .xsd file is as follows:

(TBD)

The GENSO XML messages are sent as a stream of characters, via the Internet, over a TCP/IP connection. The port and the IP address are found in the GSSL for the GSS nodes and in the ASL for the AS nodes. All messages sent in the GENSO network, are initiated by the clients, and since the MCC acts as client for both the GSS and the AS, all communication with other nodes are initiated by the MCC. The connection will be initiated by the MCC, the XML message will be sent, and the connection will be held until a response is received, or a timeout is reached.

1.1.17 The Message Handler module shall listen for, and accept incoming MCC module GENSO XML messages. These messages shall be sent over a TCP/IP connection on port 6124 (TBD standard port) or a user specified port, as a stream of characters.

1.1.18 The Message Handler module shall send outgoing MCC module GENSO XML messages. These messages shall be sent as a stream of characters.

These requirements relates to user level requirement 1.72, stating that all software modules in the MCC shall communicate in a human readable format. The messages exchanged are defined the same way as the messages sent between the nodes in the network.

10.2.2 Descriptions and diagrams

To describe the design of the Message Handler module, two types of UML diagrams are used. An activity diagram to describe the flow of the module, and a class diagram to show the structure of the module. First the activity diagram in Figure 10.11 on the next page is described.

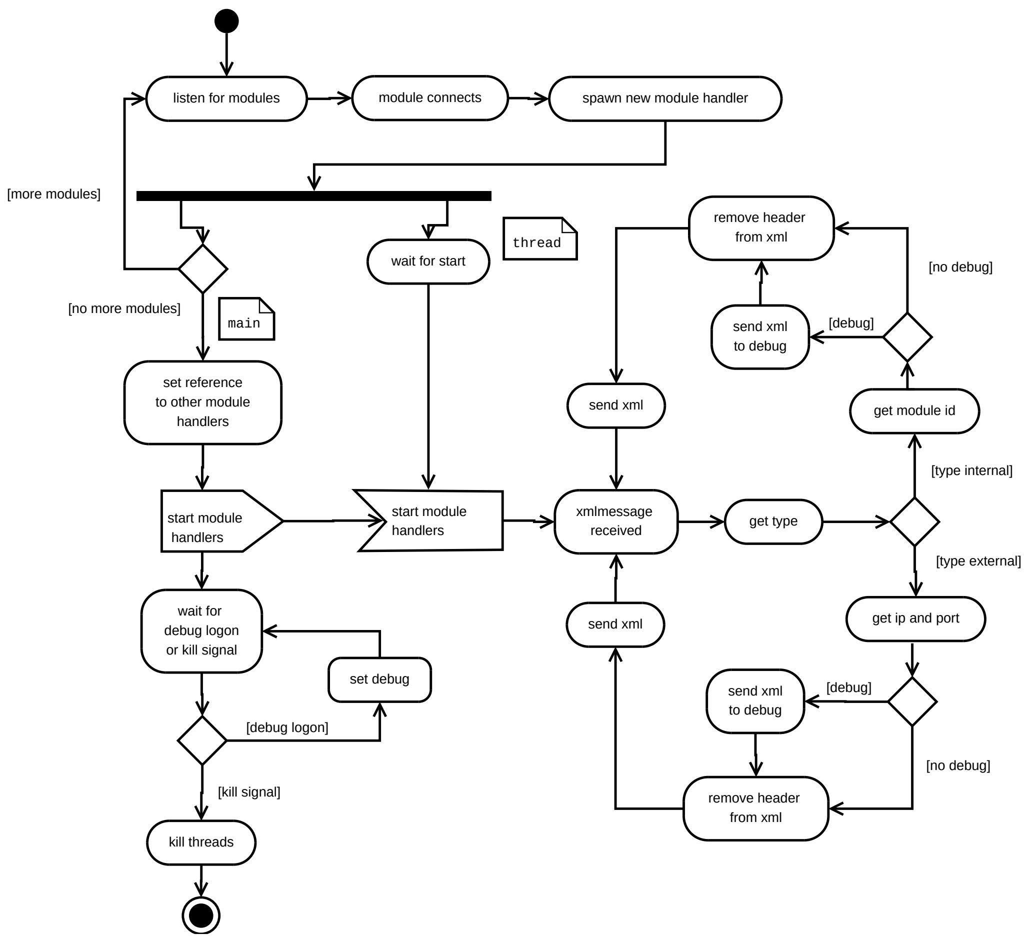


Figure 10.11: The figure shows the activity diagram for the Message Handler module.

The program flow is as follows:

1. The Message Handler module starts, and listens for connections from other modules
2. When a module connects, a `ModuleHandler` thread is created with connection to that module. The thread is not started.
3. The two previous steps are repeated until all modules have signed in.
4. When all modules have signed in, references to all other `ModuleHandler` threads are passed to each `ModuleHandler` thread.
5. All threads are started, and messages can be exchanged.

6. The `ModuleHandler` threads enters a loop where they receive, handle and sends messages based upon the headers.

7. If someone logs on to the debug channel, a copy of all messages with headers will be sent to that connection.

8. The Message Handler module flow ends when the program is killed.

This program flow has to be designed object oriented, and this design is represented with a class diagram in Figure 10.12:

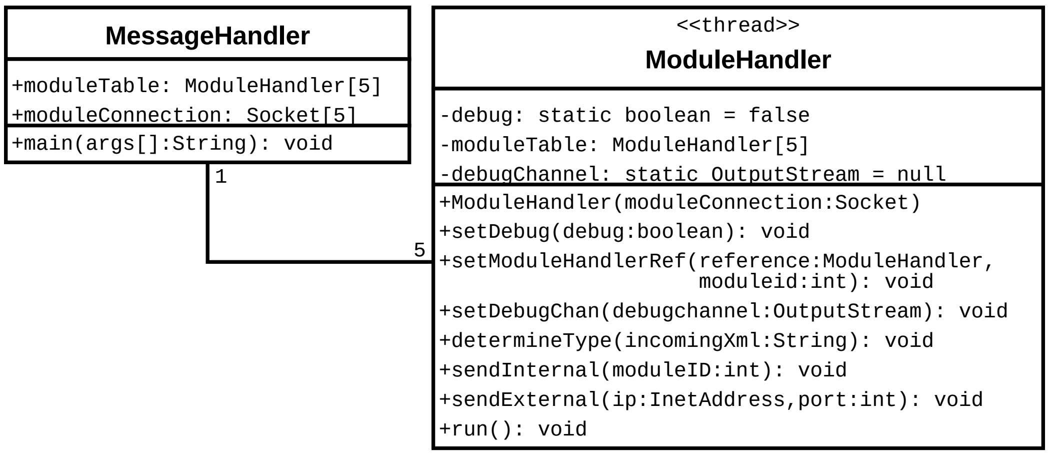

MessageHandler
+moduleTable: ModuleHandler[5]
+moduleConnection: Socket[5]
+main(args[]:String): void
1
5
<<thread>>
ModuleHandler
-debug: static boolean = false
-moduleTable: ModuleHandler[5]
-debugChannel: static OutputStream = null
+ModuleHandler(moduleConnection:Socket)
+setDebug(debug:boolean): void
+setModuleHandlerRef(reference:ModuleHandler, moduleid:int): void
+setDebugChan(debugchannel:OutputStream): void
+determineType(incomingXml:String): void
+sendInternal(moduleID:int): void
+sendExternal(ip:InetAddress,port:int): void
+run(): void

Figure 10.12: The figure shows the class diagram for the Message Handler module. Two classes has to be implemented. One to act as the initial Message Handler application, spawning the threads needed to handle communication to the other modules.

The first class in the Message Handler module, is the `MessageHandler` class, used to start the Message Handler module. This class only consists of a main method, used to invoke instances of `ModuleHandler` when another module signs in to the Message Handler module.

The `ModuleHandler` class is extended from the thread class, to make all instances of it run as a thread. The `ModuleHandler` threads are spawned when a module connects to the Message Handler module, and used to control the message connection to each module. A `ModuleHandler` thread listens for incoming GENSO XML messages, and takes care of sending them to the right receiver. The receiver can be either another module, or another node in the network, and this difference has to be handled.

The method `sendInternal` is used to send XML to other modules. Since there are 5 other modules to send messages to, a module ID has to be provided. Each module must provide a module ID when they log on to the Message Handler module. The module ID is an integer from 0 to 4 and the mapping is as follows:

Module name:	moduleID integer:
Scheduler module	0
Predict module	1
Data Handler module	2
User Interface module	3
Database module	4

When a connection is established to the Message Handler module, the first object received must be an integer defining the module ID, otherwise the connection must be shut down and has to be started again.

The method `sendExternal` (see Figure 10.12 on the preceding page) is used to send messages to another node on the network. Since no persistent connection exists to other nodes, one has to be started each time a message has to be sent. This requires that the IP address and port number is specified. When the connection has been made, and the GENSO XML message has been sent, the connection has to be kept alive to wait for a message reply or as long as the timeout period defines.

The method `determineType` (see Figure 10.12 on the facing page) is used to figure out if the message is of internal or external type. This is read from the XML message as the header information (see section 10.2.3 on the next page), put there by the sending module. The header must also contain information about the receiver of the message. If type is external, the IP and port number, and if type is internal a module id.

At last the method `setModuleHandlerRef` (see Figure 10.12 on the facing page) is used to pass references to all other modules connected to the Message Handler module, to the `ModuleHandler` threads. This information will be saved as stated in table 10.2.2, in the array of `ModuleHandler`. That way each `ModuleHandler` thread can lookup which thread to send a message to.

The attribute called `debug` (see Figure 10.12 on the facing page) is used to determine whether someone has connected to get all the messages that is exchanged in the Message Handler module. The variable is static, which makes it possible to change the value of it in all instances of `ModuleHandler` at once. That way the `sendInternal` and `sendExternal` methods can check for debug, and send messages to the debug connection if it is present. This is done via the static attribute `debugchannel`(see Figure 10.12 on the preceding page) which holds the output stream to the debug viewer.

10.2.3 Interface

The command interface to the Message Handler module is defined as the following XML messages:

Listing 10.1: Example of a XML message sent to a MCC module

```
<GENSOmsg>
  <msgHeader>
    <type>internal</type>
    <moduleID>2<moduleID>
  </msgHeader>
  <msgPayload>
     ...
  </msgPayload>
</GENSOmsg>
```

Listing 10.2: Example of a XML message sent to another node in the GENSO network

```
<GENSOmsg>
  <msgHeader>
    <type>external</type>
    <IP>86.58.135.161</IP>
    <port></port>
  </msgHeader>
  <msgPayload>
     ...
  </msgPayload>
</GENSOmsg>
```

The other interfaces that the Message Handler module provides, are defined as stream connections over TCP/IP.

Modules in the MCC must implement the following:

TCP/IP connection on port 6124 and IP address of Message Handler
Outgoing character stream for sending XML
Incoming character stream for receiving XML
First character send must be a integer number stating module ID

Other nodes in the network must implement the following:

Accept TCP/IP connection on IP address and port specified in either GSSL or ASL
Incoming character stream for receiving XML
Outgoing character stream for sending XML

The debug application can connect, after the modules, on a TCP/IP connection on port 6124, and read all XML messages sent as a stream of characters.

10.3 Scheduler

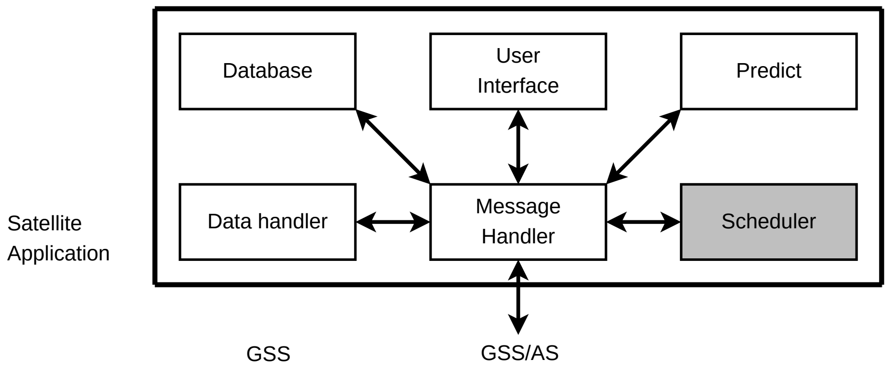


Figure 10.13: The modules of the MCC. Current module is the Scheduler module. As shown The Scheduler module communicates with all the others modules using XML messages.

The Scheduler module is the module responsible of handling the planning and booking of a communication session. Furthermore the module is responsible for watching for upcoming confirmed passes and initiate the communication session. As shown in Figure 10.13 the Scheduler module interacts with all the other modules using XML.

10.3.1 Requirements

To subtract the system level requirements to the module the user level requirements regarding scheduling have to be identified. This lead in into the user requirements defined in table 10.1

1.11	1.12	1.14	1.21	1.22	1.33	1.32

Table 10.1: The user level requirements that inflict on the development on the scheduler module

The system level requirements can then be derived:

1.1.30 The MCC shall upon request of a flight plan calculation request and receive the GSSL from the AS.
The GSSL is formed as a GENSO XML message containing the following informations of each GSS:

Variable name:	Type name:
GSS id	String
IP	String
Port	int
Location	string
Latitude	float
Longitude	float
(TBD)	

To fulfill user requirement 1.11 the Scheduler module shall be able to predict which ground stations the satellite passes. Therefore the Scheduler module needs to know which ground stations there are available in the network. This is done by asking the AS for an updated list of GSSes in the network.

1.1.32 The MCC shall book each pass, selected by the user in the flight plan, at the regarding GSS. The booking request is a GENSO XML containing:

Variable name:	Type name:
booking id	int
MCC id	String
GSS id	String
Satellite id	String
Start time for the reservation	UNIX time stamp
End time for the reservation	UNIX time stamp
Confirmation status of the reservation	boolean
Received by GSS status	boolean

As specified in user requirement 1.21 the user shall be able to book the selected GSS, therefore the Scheduler module makes a booking request to each GSS the user wishes to book.

1.1.33 The MCC shall request capabilities from each GSS in the flight plan
The capabilities are:

– Frequency band(s) - int array (TBD)

– Modulation type(s) - String array (TBD)
– (TBD)

As specified in user requirement 1.14 the user shall be able to view the capabilities of each GSS. Therefore the Scheduler module request capabilities of each GSS in the flight plan calculated by the Predict module.

1.1.34 The calculated flight plan and the capabilities of each GSS shall be returned to the User Interface module The flight plan is a GENSO XML message containing:

Variable name:	Type name:
GSS ID	String
GSS location	String
Pass start time	UNIX time stamp
Pass end time	UNIX time stamp
GSS location coordinates	float array

The capabilities are defined in system level requirement 1.1.33
User level requirement 1.12 applies that the flight plan constructed in requirement 1.1.38 and the capabilities described in requirement 1.1.33 shall be parsed to the User Interface module.

1.1.35 After a reservation of a pass the MCC must await a confirmation/denial status from the GSS and write the status and the booking informations into the Database module. If the connection times out the status will be registered as denied. The status is the same XML message as used in the booking request with the "confirmation status of the reservation" field updated. The content of the XML message is described in requirement 1.1.32

The user requirement 1.22 specifies that the status of a booking shall be visible to the user, in order to achieve that, the Scheduler module registers the status from each GSS, the information is stored in the database.

1.1.36 The Scheduler module must poll the Database module every minute (TBD) to monitor if there is a upcoming pass within the next 5 minutes (TBD). If this is the case the Scheduler module sends an XML message to the Data Handler module to initiate the connection. The XML message contains:

– GSS IP address String
– GSS connection port int

User requirements 1.32 and 1.33 specifies that a port shall be opened when a communication session begins. The Scheduler module polls the Database module to check when it is time to initiate a pass. When it is time the Scheduler module activate the Data Handler module.

1.1.37 The Scheduler module uses the Predict module to calculate the flight plan. The Scheduler module sends a GENSO XML message to the Predict module containing:

Variable name:	Type name:
Satellite ID	String
GSSL	TBD
Start time	UNIX time stamp
End time	UNIX time stamp

The Scheduler module receives a GENSO XML message with the following result from the Predict module in GENSO XML:

Variable name:	Type name:
GSS ID	String
Pass start time	UNIX time stamp
Pass end time	UNIX time stamp

To fulfill user requirement 1.11 the GSSes in the satellite path in a given period has to be found. The Predict module is used to do that.

1.1.38 When the Predict module has calculated the GSSes in the satellite path, the Scheduler module must ask every GSS for the passover time of the satellite, and use that information to create the flight plan.

The information about available passes in user requirement 1.12 is provided by requesting a list of passes from each GSS that is calculated as being in the satellite path. In that way the information about available passes shown to the user, is only containing calculated informations from the selected GSS. In that way the Predict module is only used to filter the amount of GSSes to ask for passes.

1.1.39 After each pass, a pass report shall be requested and received by the Scheduler module. The pass report is a GENSO XML message containing:

Variable name:	Type name:
Booking id	int
Packet count	int
Start time of the pass	UNIX timestamp
End time of the pass	UNIX timestamp

The Scheduler module must after each ended communication session send a request to the GSS to receive a pass report.

10.3.2 Descriptions and diagrams

After the definition of the requirements a derivation of the design is performed. This is done by identifying the flow in the module and identifying the services the module offers to other modules, as well as the services the module uses in other modules.

The flow is presented in three activity diagrams

- Construct flight plan (Figure 10.14 on the facing page)
- Book GSS (Figure 10.15 on page 78)
- Monitor upcoming satellite passes (Figure 10.16 on page 79)

Construct flight plan

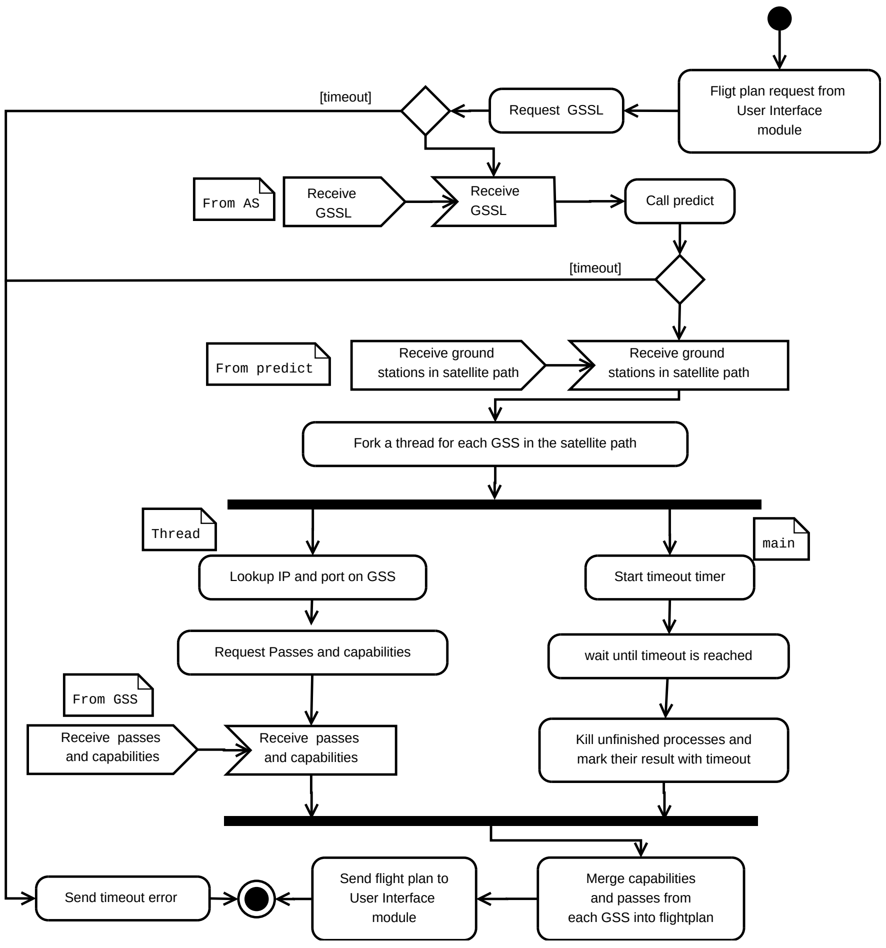

Fligt plan request from User Interface module
Request GSSL
[timeout]
From AS
Receive GSSL
Receive GSSL
Call predict
[timeout]
From predict
Receive ground stations in satellite path
Receive ground stations in satellite path
Fork a thread for each GSS in the satellite path
Thread
main
Lookup IP and port on GSS
Start timeout timer
Request Passes and capabilities
wait until timeout is reached
From GSS
Receive passes and capabilities
Receive passes and capabilities
Kill unfinished processes and mark their result with timeout
Send timeout error
Send flight plan to User Interface module
Merge capabilities and passes from each GSS into flightplan

Figure 10.14: UML activity diagram showing the flow of constructing a flight plan in the Scheduler module

The program flow of constructing a flight plan is as follows:

1. A flight plan is requested by the User Interface module
2. The GSSL is requested and received from the AS. If the AS does not respond, a timeout error is send to the User Interface module
3. The Predict module is called with the GSSes in the GSSL and a response containing the GSSes in the satellite path is returned. If the Predict module does not respond, a timeout error is send to the User Interface module
4. The Scheduler module spawns a thread for each GSS in the predicted satellite path and look up the IP and port of the GSS.
5. Each thread establishes connection to a GSS and request a prediction of passes and the

capabilities of the GSS. If a GSS does not respond within the timeout period the thread is killed and a timeout mark is appended to the GSS

6. The received satellite passes, capabilities and timeout marked GSSes is merged into one flight plan and the plan is send to the User Interface module

Book GSS

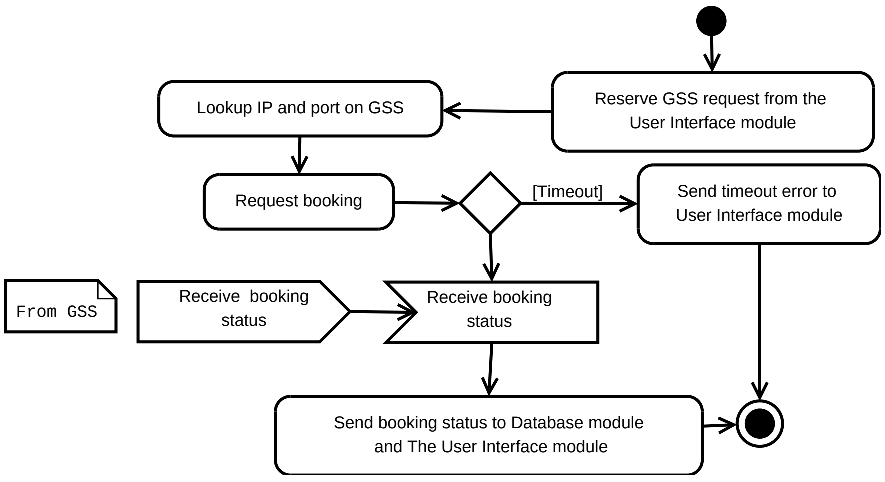

Lookup IP and port on GSS
Reserve GSS request from the User Interface module
Request booking
[Timeout]
Send timeout error to User Interface module
From GSS
Receive booking status
Receive booking status
Send booking status to Database module and The User Interface module

Figure 10.15: UML activity diagram showing the flown of a booking performed by the Scheduler module

The program flow for a booking a GSS (Figure 10.15) is as follows:

1. The Scheduler module receives a request on a booking for one satellite pass. (If the user have selected several GSSes this function is called multiple times)
2. The Scheduler module looks up the IP and port of the GSS and sends a booking request to the GSS. If the GSS does not respond the “Received by GSS status” is set to false.
3. The status of the booking is send to the Database module and the User Interface module

Monitor upcoming satellite passes

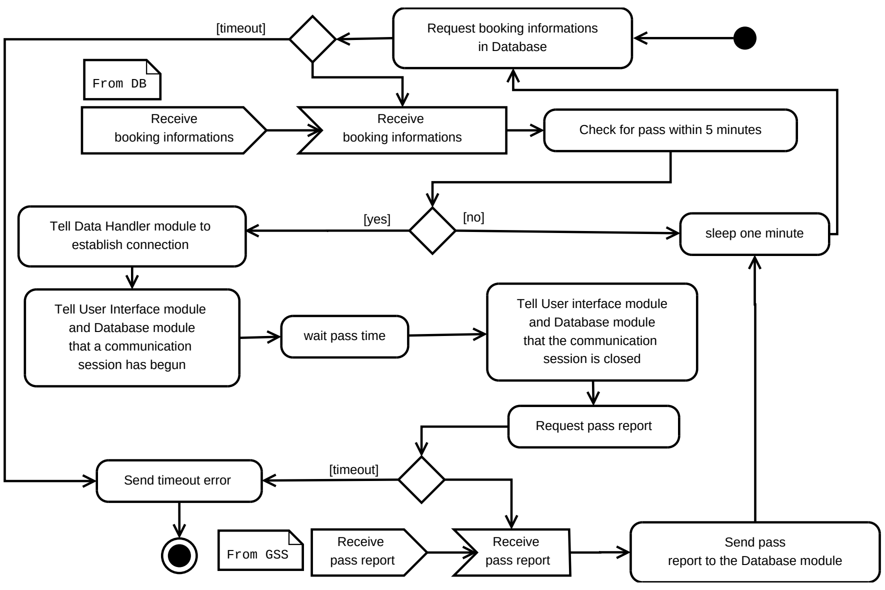

[timeout]
Request booking informations in Database
From DB
Receive booking informations
Receive booking informations
Check for pass within 5 minutes
Tell Data Handler module to establish connection
[yes]
[no]
sleep one minute
Tell User Interface module and Database module that a communication session has begun
wait pass time
Tell User interface module and Database module that the communication session is closed
Request pass report
Send timeout error
[timeout]
From GSS
Receive pass report
Receive pass report
Send pass report to the Database module

Figure 10.16: UML activity diagram showing the flow of the Scheduler module monitoring for upcoming reserved passes

The program flow for “monitoring of upcoming reserved passes” (Figure 10.16) is as follows:

1. The Scheduler module request the list of upcoming reserved passes from the Database module. If the Database module does not respond, a timeout error is send to the User Interface module

2. The Scheduler module checks if there are any passes within the next five minutes, if not it sleeps one minute

3. If there are any upcoming passes within the next five minutes, the Data Handler module is told to initiate a communication session with the regarding GSS.

4. The User Interface module module and the Database module is told that a communication session is started

5. The Scheduler module waits the period of the pass

6. The User Interface module and the Database module is told that the session is over and a pass report is requested from the GSS. If the GSS does not respond a timeout error is send to the User Interface module

7. Finally the pass report is sent to the Database module

Services of the module

To ease the design of the Scheduler module the services of the module are identified. The Scheduler modules receive the following informations and commands from other modules and external nodes:

1. A GSSL from the AS
2. Predicted GSSes in the satellite path from the Predict module
3. Passes of the satellite on a specific GSS and the capabilities of the GSS from the specific GSS
4. Booking informations containing confirmed satellite passes from the Database module
5. Pass report from a GSS
6. Booking status telling if the booking is confirmed or not, from the GSS
7. A flight plan request from the User Interface module
8. A booking request from the User Interface module

The Scheduler module sends the following information to the other modules or nodes

1. A flight plan containing the GSSes that the satellite passes, the time window of each pass and the capabilities of the GSSes in the satellite pass. This is sent to the User Interface module.
2. A message to the Message Handler module to start a communication session.
3. A message to the User Interface module and Database module that a communication session has started.
4. A message to the User Interface module and Database module that a communication session has ended.
5. A pass report to the Database module.
6. The status of a requested booking .

Class diagram of the Scheduler module

The flow in the Scheduler module leads into the determination of the class diagram in figure 10.17

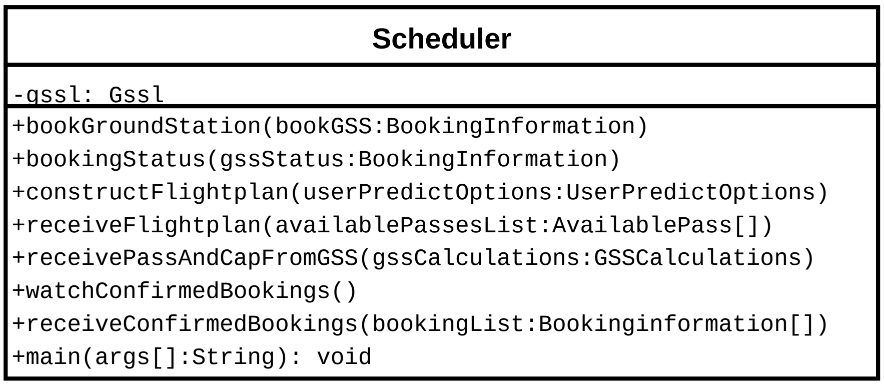

Scheduler
-gssl: Gssl
+bookGroundStation(bookGSS:BookingInformation)
+bookingStatus(gssStatus:BookingInformation)
+constructFlightplan(userPredictOptions:UserPredictOptions)
+receiveFlightplan(availablePassesList:AvailablePass[])
+receivePassAndCapFromGSS(gssCalculations:GSSCalculations)
+watchConfirmedBookings()
+receiveConfirmedBookings(bookingList:Bookinginformation[])
+main(args[]:String): void

Figure 10.17: UML Class diagram showing the content of the Scheduler module pack

The class diagram contains methods solving the three flows, the incoming information is handled in separate methods in order to let an incoming message handler call a function matching the incoming message.

10.3.3 Interfaces

The Scheduler module interface description handles the interfaces to the Scheduler module, meaning the XML messages that is received from external nodes and modules. The messages send to the module is derived in "service of the module" and listed in the same order:

Listing 10.3: The GSSL received from the AS

```
...
 <GSS>
    <GSSID></GSSID>
    <IP></IP>
    <port></port>
    <location></location>
    <Latitude></Latitude>
    <Longitude></Longitude>
     ...
 </GSS>
...
</GSSL>
```

Listing 10.4: Predicted GSSes in the satellite path from the Predict module

```
...
 <AvailablePass>
    <groundStationId></groundStationId>
    <satelliteId></satelliteId>
    <startTime></startTime>
    <endTime></endTime>
 </AvailablePass>
...
</availablePassesList>
```

Listing 10.5: Passes of the satellite on a specific GSS and the capabilities of the GSS from the specific GSS

```
    <groundStationId></groundStationId>
    <satelliteId></satelliteId>
    <startTime></startTime>
    <endTime></endTime>
    <capabilities></capabilities>
<GSSCalculations>
```

Listing 10.6: Booking informations containing confirmed satellite passes from the Database module

```
...
 <BookingInformation>
    <bookingID></bookingID>
    <gssID></gssID>
    <mccID></mccID>
    <satID></satID>
    <startTime></startTime>
    <endTime></endTime>
    <receivedByGSS></receivedByGSS>
    <timeslotConfirmed></timeslotConfirmed>
 </BookingInformation>
...
</bookingList>
```

Listing 10.7: Pass report from a GSS

```
<passreport>
    <bookingID></bookingID>
    <startTime></startTime>
    <endTime></endTime>
</passreport>
```

Listing 10.8: A flight plan request from the User Interface module

```
<UserPredictOptions>
    <satID></satID>
    <startTime></startTime>
    <endTime></endTime>
</UserPredictOptions>
```

Listing 10.9: Booking status telling if the booking is confirmed or not, from the GSS

```
<BookingInformation>
    <bookingID></bookingID>
    <gssID></gssID>
    <mccID></mccID>
    <satID></satID>
    <startTime></startTime>
    <endTime></endTime>
    <receivedByGSS></receivedByGSS>
    <timeslotConfirmed></timeslotConfirmed>
</BookingInformation>
```

Listing 10.10: A booking request from the User Interface module

```
<BookingInformation>
    <bookingID></bookingID>
    <gssID></gssID>
    <mccID></mccID>
    <satID></satID>
    <startTime></startTime>
    <endTime></endTime>
    <receivedByGSS></receivedByGSS>
    <timeslotConfirmed></timeslotConfirmed>
</BookingInformation>
```

This concludes the interfaces of the Scheduler module. The next module to be described is the Predict module.

10.4 Predict

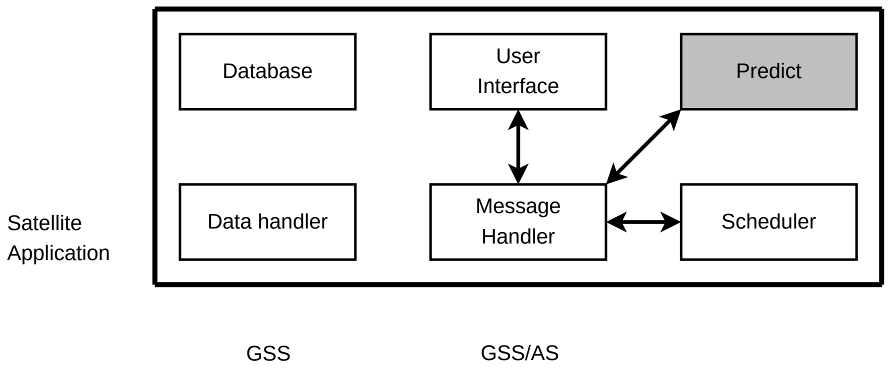


Figure 10.18: The modules of the MCC. Current module is Predict. As shown the Predict module communicates with the User Interface module and the Scheduler module using XML messages.

The main purpose of the Predict module is to offer an interface for the Scheduler module, to calculate satellite passes for a given satellite over a given ground station. These passes are defined by the specific satellites path around the Earth. The background of Predict is described in Appendix A on page 105. It is possible to calculate the satellite's position at a given time from the orbit. John A. Magliacane, KD2BD has developed a software library to support these special needs. Figure 10.18 shows the Predict module in the MCC.

10.4.1 Usage of Predict library

Predict is an open-source, multi-user satellite tracking and orbital prediction program written under the Linux operating system[8]. This means the program is able to calculate a satellite position, from a given set of keplerian elements. This set of elements are contained in Two Line Element set (TLE), which is maintained by CelesTrak (part of the Center for Space Standards and Innovation (CSSI)).

The TLE is a mathematical description of a satellite orbit, including launch year and satellite name. An example of a NORAD TLE could be:

```
Line 0: AAU CUBESAT
Line 1: 1 27846U 03031G   06333.55058209  .00000050  00000-0  43268-4 0  8499
Line 2: 2 27846  98.7185 340.4346 0008797 291.2307  68.7941 14.20848903177184
```

Line 0 is a descriptive and unique name of the satellite. Line 1 and 2 is the standard TLE, with an integrated checksum. Each column is described in appendix B on page 109.

Like a TLE, each ground station also needs a location specification. This is handled with a QTH file in Predict. QTH is a code representing a location in radio telegraphic language. The location description is a set of latitude, longitude and altitude coordinates representing the placement of the ground station antenna. The QTH file also includes a call sign or name of the ground station for identification. An example of a QTH-file could be:

```
Line 0: AAUGND
Line 1: 57.1
Line 2: −9.85
Line 3: 13
```

10.4.2 Requirements

From user level requirements 1.11 and 1.12, the following system level requirements is derived, which specifies the functionality of the Predict module.

1.1.45 To predict the passes of the host spacecraft over the stations in the network the MCC needs to know:

- Satellite ID (32 Character CelesTrack string)
- Ground Station List
- Start and end time of the time window in UNIX timestamp

This describes the basic information needed to execute the Predict library. The timestamps are needed to get all possible passes within the time window. This is achieved by calling the Predict library with the start time, which returns the first pass. The end time of the first pass is then used to predict the next pass. This loop continues until the end time specified by the user has been reached.

1.1.46 Predict must calculate the spacecraft path in coordinates, supporting the availability to draw the path on a map at a given time window. The time window must be specified in UNIX timestamp, by a start and an end timestamp.

In order to draw the map, a time window is needed. The satellite ID will refer to the TLE set produced by Celestrack.

10.4.3 Description and diagrams

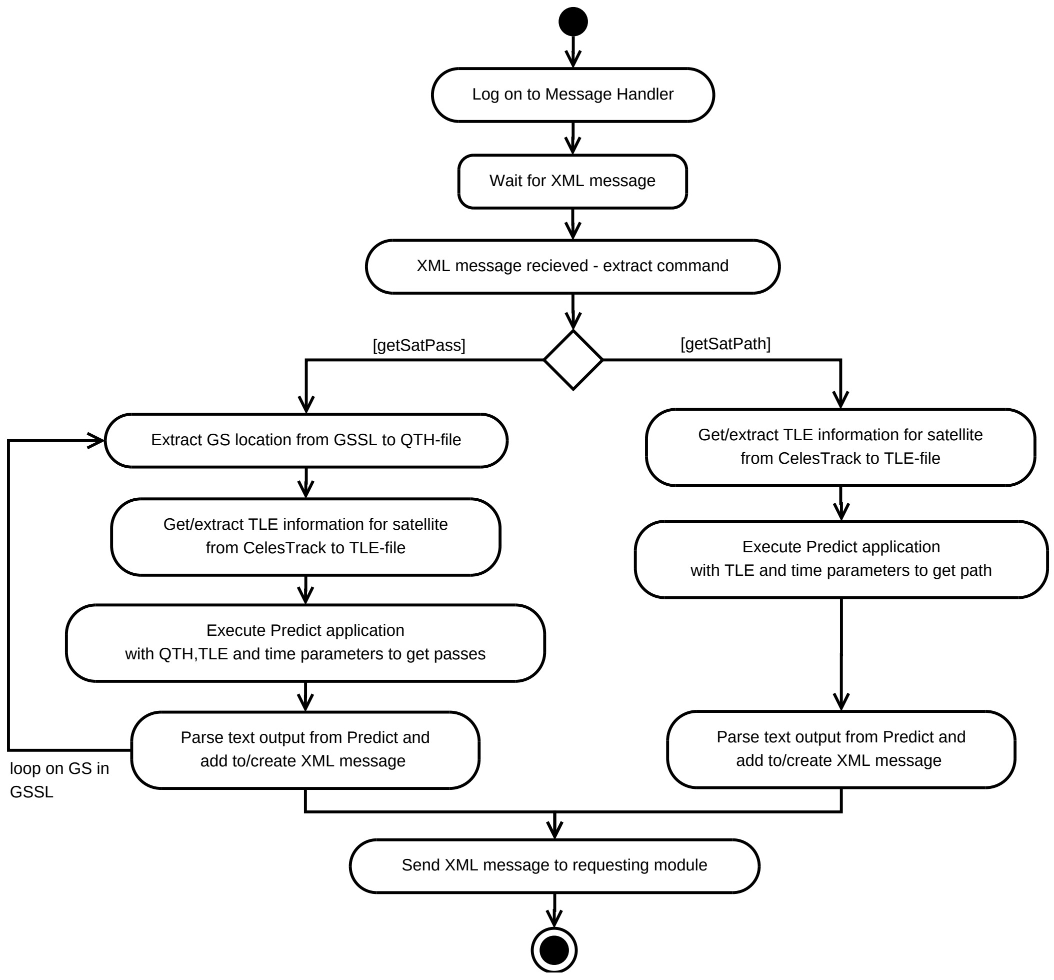

Log on to Message Handler
Wait for XML message
XML message recieved - extract command
[getSatPass]
[getSatPath]
Extract GS location from GSSL to QTH-file
Get/extract TLE information for satellite from CelesTrack to TLE-file
Execute Predict application with QTH,TLE and time parameters to get passes
Parse text output from Predict and add to/create XML message
loop on GS in GSSL
Get/extract TLE information for satellite from CelesTrack to TLE-file
Execute Predict application with TLE and time parameters to get path
Parse text output from Predict and add to/create XML message
Send XML message to requesting module

Figure 10.19: UML Activity diagram specifying Predict module flow

The UML activity diagram (see figure 10.19) shows the flow of the module. There are two possible ways through the module.

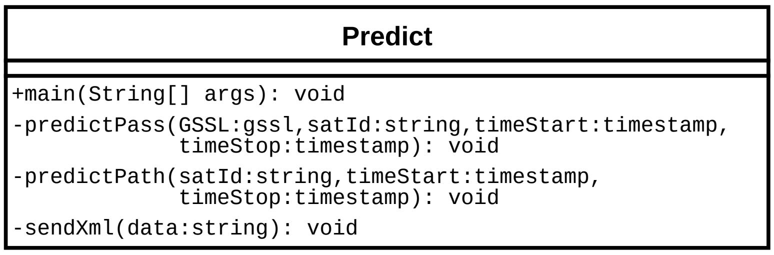
getSatPass flow The Scheduler module passes a XML message to Predict. The Predict module will then receive the message, parse it and extract the command inside the XML message. If the command is `getSatPass`, a loop is entered, in which the ground station and satellite information will be extracted.

The Predict library needs TLE and QTH information as files, therefore is it necessary to create these files before executing the library. When these files are created, an execution of the library will take place, including the time information given by the content of the message.

After execution the output from the Predict application is parsed and put in the XML message specified by the interface description. The loop then continues and the next available ground station will produce a new element in the passes list.

getSatPath flow When receiving a `getSatPath` message, the Predict library only needs the TLE of the satellite and a time window. Then Predict is executed once to get the satellite path. The returning list with coordinates of the satellite has to be parsed into XML before sending the message in return.

With the requirements and flow in mind, it is possible to construct the UML Class diagram for Predict class. External communication will go through the interfaces which is all ready described.



```
Predict

+main(String[] args): void
-predictPass(GSSL:gssl,satId:string,timeStart:timestamp,
             timeStop:timestamp): void
-predictPath(satId:string,timeStart:timestamp,
             timeStop:timestamp): void
-sendXml(data:string): void
```

Figure 10.20: UML Class diagram for Predict module

The UML class diagram shows the 4 methods inside the module. Each Predict method will be invoked by the main method, and the sendXml method will handle communication between the Predict module and the Message Handler module, by sending requested information back to the requesting module.

10.4.4 Interfaces

The Predict module interfaces with the Scheduler module by providing a list of predicted passes for the time window specified by the request. The Predict module interacts with the User Interface module as well, providing a coordinate list telling where the satellite is at a given time. This will give the User Interface module the ability to draw the entire path of the satellite orbiting the Earth. All interfaces shall be implemented in XML.

- The Scheduler module will have to provide an XML message including the satellite ID as a string, a GSSL and a time window specified by start and end time in UNIX timestamp. The following XML example will define the interface:

```
<predictRequest>
  <command>getSatPass</command>
  <satelliteId></satelliteId>
  <startTime></startTime>
  <endTime></endTime>
  <gsslList>
     ...
    <gsslItem>
      <Id></Id>
      <Latitude></Latitude>
      <Longitude></Longitude>
      <Altitude></Altitude>
    </gsslItem>
     ...
  </gsslList>
</predictRequest>
```

- The User Interface module will have to provide an XML message including the satellite ID as a string and the time window in UNIX timestamp. Whenever the request is processed, the Predict module will return an XML message to the requesting module. The request has to fulfill the following XML specification:

```
<predictRequest>
  <command>getSatPath</command>
  <satelliteId></satelliteId>
  <startTime></startTime>
  <endTime></endTime>
</predictRequest>
```

10.4.5 Test specification

It will be assumed in the test that the Predict library is tested properly by the developers of the library. This test will be based on comparing output from the Predict application calculations with the output from the Predict module. It is then possible to test whether the Predict module can calculate one pass correctly. The test must send an XML message to the Predict module containing information connected to a known TLE and QTH file. The exact same informations must be provided to the Predict application as well. The two outputs can then be compared.

The same procedure must be performed for a set of ground stations in a GSSL, to test whether the Predict module is able to find a set of ground stations which the satellite will pass. At least one of the known ground stations must be out of range in the given time window. The returned XMl message must not contain the ground station being out of range.

10.5 Database

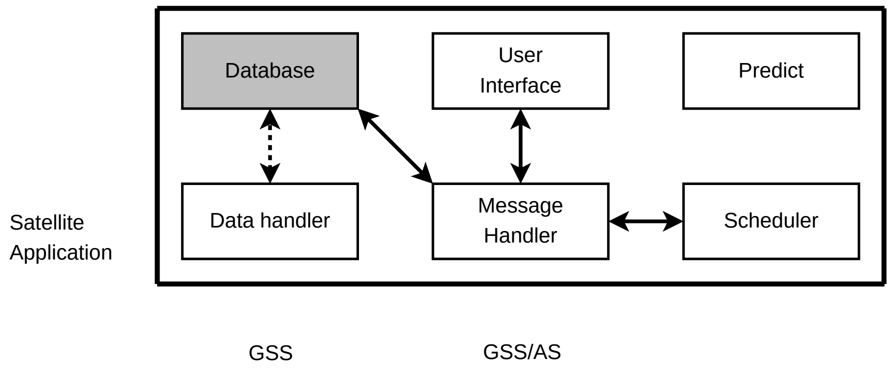


Figure 10.21: The Figure shows the Database module and the modules the Database module interfaces with. The Database module communicates with the Scheduler and User Interface module. During a satellite transmission the Database module receives satellite mission data from Data Handler module as well.

Various types of data are exchanged between the nodes of the GENSO network, as well as between the different modules in the MCC. The user level requirements specifies that some of these data types has to be stored in a database. On the MCC this is handled by the Database Module, shown in Figure 10.21, and the possible data types are:

- a booking information
- a pass report
- a satellite mission data packet

For each data type the Database module has to provide the following functionality:

- Store the data type in the database
- Retrieve the data present in the database and send it to the requesting module
- Change the values of stored data types in the database

10.5.1 Requirements

User level requirements 1.21, 1.22, 1.34, 1.35 and 1.37 (see chapter 7 on page 37) have specifications involving the Database module. To comply and fulfill these user level requirements, it has been necessary to make a set of system level requirements. These system level requirements will now be described one by one with reference to the matching user level requirement.

1.1.60 A satellite mission data packet consists of the following variable types:

Variable name:	Type name:
spacecraft id	String
direction	boolean
reception / transmission node id	String
reception / transmission time (UTC)	UNIX timestamp
link budget estimation (TBD)	int
Doppler-correction factor	float
S-meter value (if available)	int
transmitted data	byte

The format of a satellite mission data packet is specified in user level requirement 1.34.

As long as the above format is used, the Database module knows what to expect to receive from the Data Handler module. User level requirement 1.34 specifies that uplink and downlink data from satellite communication sessions has to be stored in the database. The Data Handler module is responsible for the communication between the user and the GSS. This module then has to forward each satellite mission data packet to the Database module.

1.1.61 A booking shall be stored in a database. The variables in a booking uses the following variable types:

Variable name:	Type name:
booking ID	int
MCC ID	String
GSS ID	String
Satellite ID	string
Start time for the reservation (UTC)	UNIX timestamp
End time for the reservation (UTC)	UNIX timestamp
Confirmation status of the reservation	int

User level requirement 1.21 specifies that the user has to be able to book a previously selected flight plan. This booking has to be stored in the database and as such it is necessary to specify the variable types used to represent a booking. The format of a booking is defined in user level requirement 1.21.

1.1.62 The Database module shall retrieve bookings from the database when requested by the User Interface module.

The User Interface module needs to show booking informations to the user. These booking informations have to be retrieved from the database by the Database module and then sent to the User Interface module. The format of these messages is specified in user level requirement 1.21 and the variables used to represent a booking are defined in system level requirement 1.1.61.

1.1.63 The pass report shall be stored in the database. The following variable types are used to represent a pass report:

Variable name:	Variable type:
booking ID	int
packet count	int
start time (UTC)	UNIX timestamp
end time (UTC)	UNIX timestamp

The format of a pass report is specified in user level requirement 1.35.

The GSS generates a pass report once a communication session with a satellite has ended. This pass report is then sent to the MCCs Scheduler module when requested. Once received, these reports has to be stored in the database.

1.1.64 Pass reports shall be retrieved from the database when requested. The format of a pass report is specified in user level requirement 1.35 and the variable types representing a pass report are specified in system level requirement 1.1.63.

Pass reports has to be retrieved from the database and shown to the user by the User Interface module. The pass reports sent between the User Interface module and the Database module will use the format specified in user level requirement 1.35.

Now that the requirements have been specified it is possible to construct an activity diagram and subsequently a class diagram of the Database module.

10.5.2 Descriptions and diagrams

An activity diagram of the Database module can be seen in Figure 10.22 on the following page.

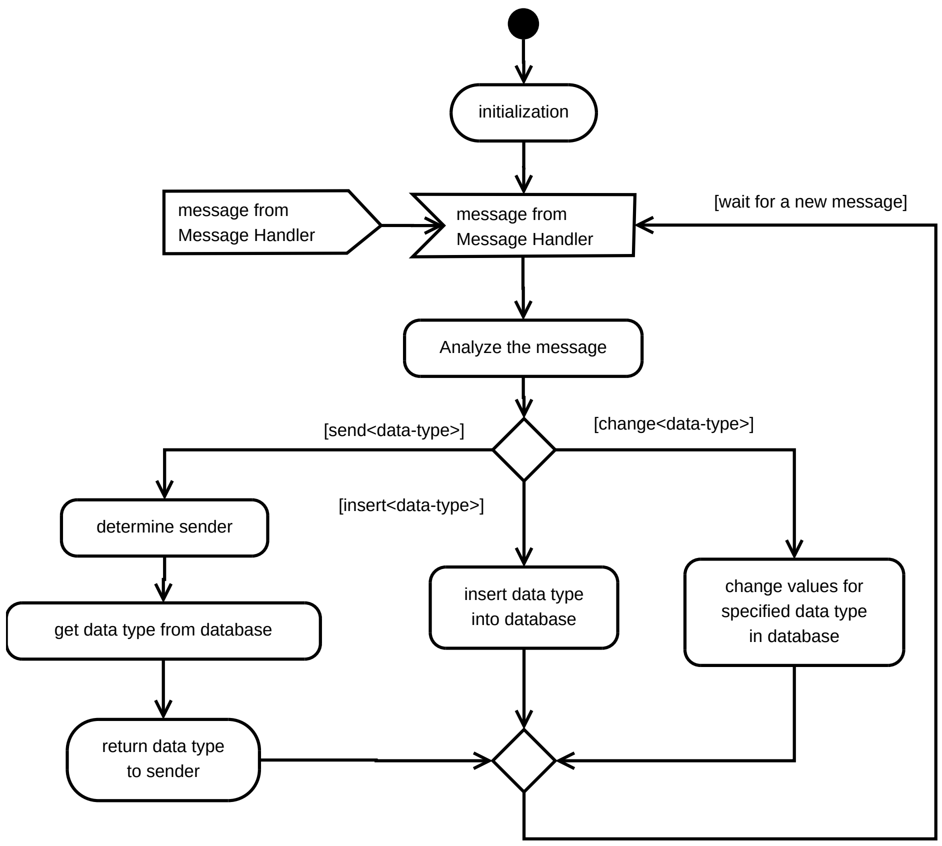

initialization
message from Message Handler
message from Message Handler
[wait for a new message]
Analyze the message
[send<data-type>]
[change<data-type>]
[insert<data-type>]
determine sender
get data type from database
return data type to sender
insert data type into database
change values for specified data type in database

Figure 10.22: The figure shows the activity diagram of the Database module.

The following steps can be seen on Figure 10.22:

1. The Database module is initialized
2. The module enters a loop where the module waits until a message is received from the other modules through the Message Handler module
3. Once a message arrives, the message is analyzed and the contents of the message determines how the module proceeds
4. If the message is an `insert` message the following occurs:
   (a) The values from the message is inserted in the database for the specified data type.
   (b) The module enters its waiting loop again
5. If the message is a `change` message the following occurs:
   (a) The values for the specified data type are changed in the database.
   (b) The module enters its waiting loop again

6. If the message is a `send` message the following occurs:
   (a) The sender of the message is determined
   (b) The specified data type is extracted from the database
   (c) The result is returned to the sender of the original message.
   (d) The module enters its waiting loop again

Now that the activity diagram has been described it is possible to construct a class diagram of the Database module (see Figure 10.23).

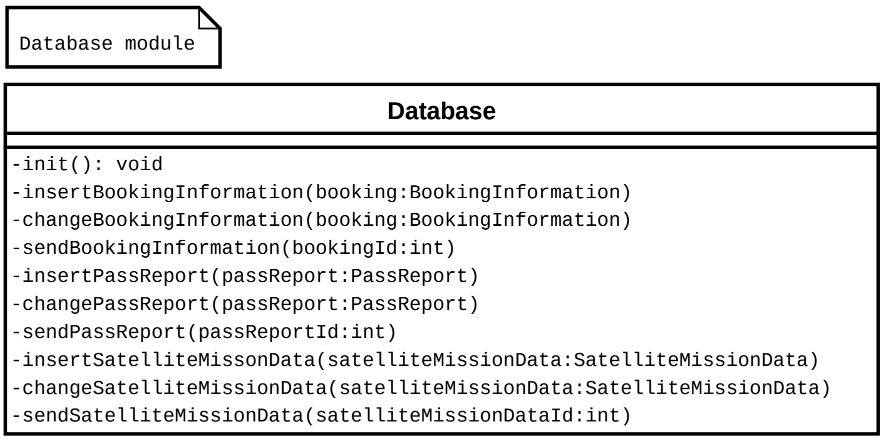

Database module
Database
-init(): void
-insertBookingInformation(booking:BookingInformation)
-changeBookingInformation(booking:BookingInformation)
-sendBookingInformation(bookingId:int)
-insertPassReport(passReport:PassReport)
-changePassReport(passReport:PassReport)
-sendPassReport(passReportId:int)
-insertSatelliteMissonData(satelliteMissionData:SatelliteMissionData)
-changeSatelliteMissionData(satelliteMissionData:SatelliteMissionData)
-sendSatelliteMissionData(satelliteMissionDataId:int)

Figure 10.23: The figure shows the class diagram of the Database module. For each data type the Database module has to handle, there are corresponding insert-, change- and get<data type>-methods.

The `init()` method is executed when the Database module is started. Configuration and the initialization is managed by this method. The following methods in the class diagram are all concerned with the Database modules interaction with the other modules of the MCC. For each data type the Database module has to handle, there are three corresponding methods:

- insert<data-type>()
  Used to insert a given data type in the database.
- change<data-type>()
  Used to change a given data type that has already been added to the database previously.
- get<data-type>()
  Used to retrieve a given data type from the database.

where <data-type> is either a booking information, a satellite mission data packet or a pass report.

Now that the design of the Database module has been explained, it is possible to specify the interfaces of the Database module.

10.5.3 Interfaces

The Database module receives messages from the Scheduler module when e.g. a booking information needs to be inserted in the database. A message that has to do with booking information will contain the following:

```
<BookingInformation>
  <bookingID></bookingID>
  <gssID></gssID>
  <mccID></mccID>
  <satID></satID>
  <startTime></startTime>
  <endTime></endTime>
  <receivedByGSS></receivedByGSS>
  <timeslotConfirmed></timeslotConfirmed>
</BookingInformation>
```

If the message deals with a satellite mission data packet from the Data Handler module the message will contain:

```
<SatelliteMissionData>
  <spacecraftID></spacecraftID>
  <direction></direction>
  <rtNodeID></rtNodeID>
  <rtTime></rtTime>
  <linkBudgetEstimation></linkBudgetEstimation>
  <dopplerCorrectionFactor></dopplerCorrectionFactor>
  <sMeterValue></sMeterValue>
  <transmittedData></transmittedData>
</SatelliteMissionData>
```

And if the message involves handling of a pass report, the message will contain:

```
<PassReport>
  <bookingID></bookingID>
  <packetCount></packetCount>
  <startTime></startTime>
  <endTime></endTime>
</PassReport>
```

10.5.4 Test specification

To make sure that the Database module fulfills the requirements specifications, it is necessary to perform a module test. The module will be tested by a set of scenarios, where each scenario tests part of the modules functionality. This implies that a blackbox test will be used.

The Database module interacts with the Scheduler, Message Handler and Data Handler module as well as the User Interface module. These modules will be used to test the Database module. By sending messages from the modules to the Database module and then comparing the messages XML contents with the actual data stored in the database, data consistency can be confirmed.

Testing the Database modules handling of booking informations

The first scenario tests the Database modules handling of booking information. This will be tested through the following steps:

- `insertBookingInformation()` test
  1. Create a booking information with predetermined content in the Scheduler module.
  2. The Scheduler sends the insert booking information message to the Database module. The Database module should then insert the booking information in the database once it receives the message from the Scheduler module.
  3. Use the User Interface module to control that the booking information has been inserted correctly by having the User Interface module show the previously inserted booking.
  4. If the User Interface module shows the expected output, the Database modules `insertBookingInformation()` method has passed its test.
- `changeBookingInformation()` test
  1. Change every variable in the previously used booking information except for the booking id. This has to remain unchanged.
  2. Send the booking information in an update message from the Scheduler module to the Database module.
  3. Verify that the values in the database has been changed to the new values of the booking information, by having the User Interface module display the booking information once more.
  4. If the User Interface module displays the expected output the `changeBookingInformation()` has passed its test.
- `sendBookingInformation()` test
  1. The User Interface module transmits a send booking information message, with the previously used booking id, to the Database module.
  2. If the message returned from the Database module is equal to the output that where display during the `changeBookingInformation()` test, the `sendBookingInformation()` has passed its test.

The same procedure can be used to test the Database modules handling of pass reports and data packets. The only difference being, that the Data Handler module should be used instead of the Scheduler module to test the Database modules handling of satellite mission data packet.

10.6 Data Handler

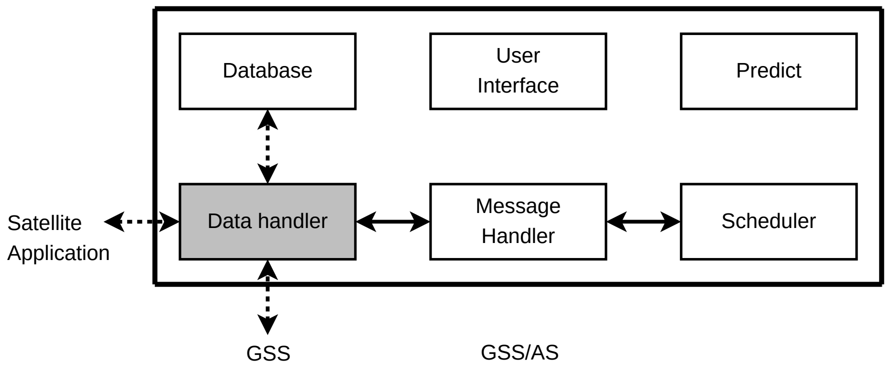


Figure 10.24: The modules of the MCC. Current module is the Data Handler module. The Data Handler module communicates with the Scheduler module and the Database module via XML messages. The Data Handler module also has connections to the Database module as well as a GSS and a Satellite Application to forward binary data.

The Data Handler module shall provide an interface for the user to connect his own Satellite Application. The purpose of the Data Handler module is to forward the satellite mission data between the the Satellite Application and the GSS while logging it in the MCC database. Because the satellite mission data must be handled binary it can not be added directly to an XML message. This implies that the Data Handler module must have direct access to the Database module and the GSS without going through the Message Handler. Figure 10.24 shows the Data Handler module in the MCC.

10.6.1 Requirements

Requirement 1.31, 1.32, 1.34 in the requirement specification in chapter 7 on page 37 relates to the Data Handler Module. To fulfill these, the following requirements are defined as the base for the design of the Data Handler Module.

1.1.75 The Data Handler module shall connect to a GSS socket on IP address and port number provided by the Scheduler

1.1.76 The Data Handler module shall provide a server socket for satellite mission data on port 6125 (TBD)

Requirement 1.31 and 1.32 states an uplink and downlink connection to the GSS. The Data Handler Module needs to know which GSS it must connect to, and when to connect. The Scheduler holds this information as it has made the reservation. The server socket is for the Satellite Application to connect to.

1.1.77 All satellite mission data shall be forwarded to the Database on port 6126 (TBD)

1.1.78 All satellite mission data shall be forwarded to the local server socket on port 6125 (TBD)

This is for logging of the satellite mission data in the database as stated in requirement 1.34. This satellite mission data must also be forwarded to the Satellite Application of the user to fulfill requirement 1.32.

1.1.79 The downlink satellite mission data shall be read 1 byte at the time (TBD) from the GSS socket

Every 1 byte (TBD) is regarded as a packet of satellite mission data for simplicity. The packet size must be a trade off between delay of buffers to fill up and overhead in the network.

10.6.2 Description and diagrams

The flow of the Data Handler Module is shown in the activity diagram in Figure 10.25 on the next page. It shows the initialization of the Data Handler Module and one pass of satellite communication. The details of the diagram are described below.

1. The Data Handler Module connects to the Message Handler module.
2. A local server socket is created waiting for the user to connect his Satellite Application.
3. When a communication session is starting, the Scheduler module provides IP address and port number to connect to.
4. Connections to the Database module and GSS are created.
5. The tasks of listening and sending to the GSS are split into two threads as read and write of satellite mission data to forward, is asynchronous.
6. The two threads each forwards the satellite mission data to the the Database byte by byte.
7. When the communication session is over, the Data Handler Module is told to close the external connections by the Scheduler.

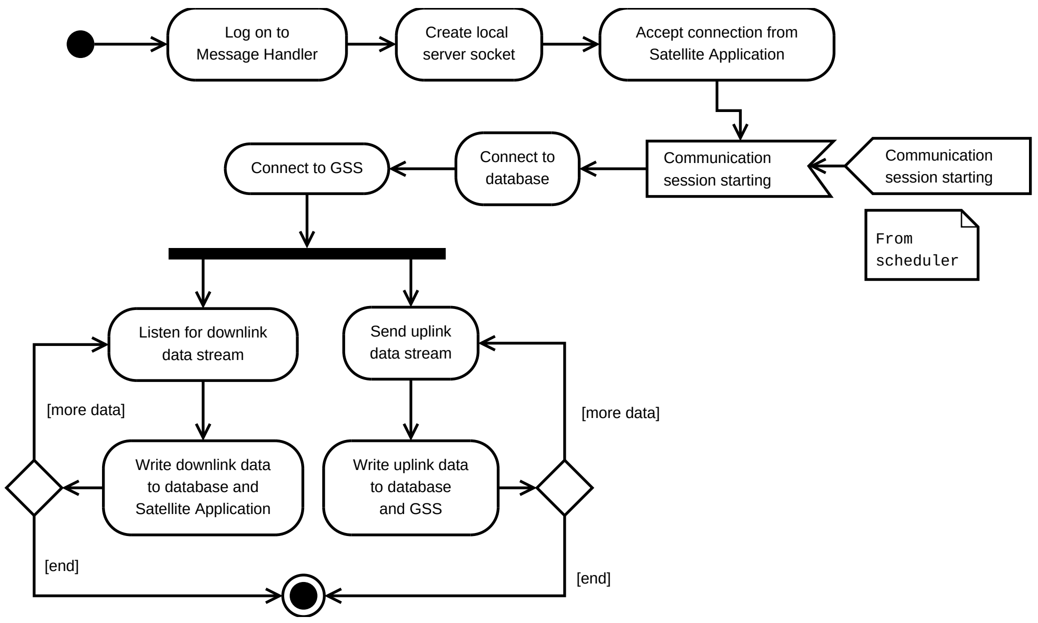

Log on to Message Handler
Create local server socket
Accept connection from Satellite Application
Communication session starting
Communication session starting
From scheduler
Connect to database
Connect to GSS
Listen for downlink data stream
Send uplink data stream
Write downlink data to database and Satellite Application
Write uplink data to database and GSS
[more data]
[more data]
[end]
[end]

Figure 10.25: Activity diagram of the Data Handler Module. It shows the initialization and one satellite communication session.

After the flow is determined a class diagram of the Data Handler Module can be constructed. This is shown in Figure 10.26. It consist of a main class `DataHandler` and a thread class `DataForwarder`. The two instances of `DataForwarder` forwards uplink and downlink satellite mission data respectively.

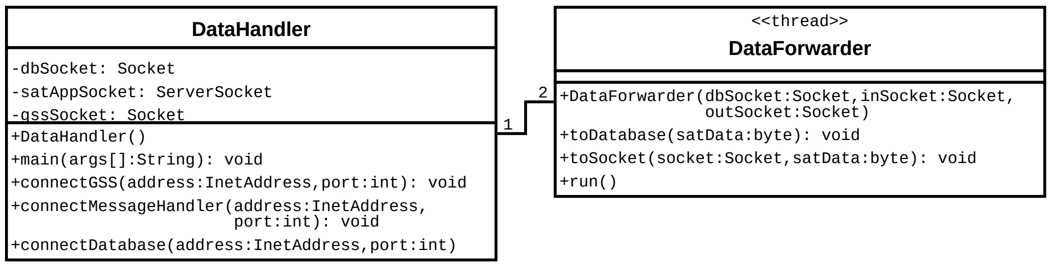

DataHandler
-dbSocket: Socket
-satAppSocket: ServerSocket
-gssSocket: Socket
+DataHandler()
+main(args[]:String): void
+connectGSS(address:InetAddress,port:int): void
+connectMessageHandler(address:InetAddress, port:int): void
+connectDatabase(address:InetAddress,port:int)
1
2
<<thread>>
DataForwarder
+DataForwarder(dbSocket:Socket,inSocket:Socket, outSocket:Socket)
+toDatabase(satData:byte): void
+toSocket(socket:Socket,satData:byte): void
+run()

Figure 10.26: UML Class diagram of the Data Handler module

10.6.3 Interfaces

The Data Handler Module is connected to the Message Handler module as well as any other internal MCC modules. Through the Message Handler module it has interface to the Scheduler module. This provides the Data Handler Module with GSS IP addresses and port numbers. The following messages is accepted:

Listing 10.11: The XML for the start messages

```
<dataHandlerRequest>
  <action>start</action>
  <IP>83.234.95.32</IP>
  <port>6128</port>
</dataHandlerRequest>
```

Listing 10.12: The XML for a stop message

```
<dataHandlerRequest>
  <action>stop</action>
  <IP>83.234.95.32</IP>
  <port>6128</port>
</dataHandlerRequest>
```

Furthermore the Data Handler Module has socket interfaces with binary data streams to the following modules and applications:

GSS
For receiving and transmitting satellite mission data (binary)

Database
For logging of all satellite mission data (binary)

Satellite Application
For the user to communicate with the satellite (binary)

10.6.4 Test specification

An implementation of the module needs to be tested to make sure that the functionality is as the stated in the requirements. The following scenario describes a module test of the Data Handler Module.

1. The Data Handler Module and implementations of Message Handler, Database and GSS is started
2. The Data Handler Module must be supplied with an input to start a session from the Message Handler module.
3. It must be verified that the Data Handler Module then provides a server socket on 6125 (TBD)
4. It must be verified that the Data Handler Module connects to the Database module on port 6126 (TBD) and the GSS on port supplied by the Message Handler module.
5. It must be verified that all data from the GSS is forwarded to the Database module and the server socket byte by byte.

6. It must be verified that all data from the server socket is forwarded to the GSS and the Database module byte by byte.

10.7 Design conclusion

The design of the MCC has now been described. General methods used throughout the design has been explained, and each module of the MCC has been described individually. As explained in chapter 6 on page 30, the supplied requirement specification by the GENSO project is still under preparation. This has also had impact on the design of the MCC.

The design in this report is not complete. As it can been seen in the requirements specification, several items needs to be either confirmed (TBC) or to be determined (TBD). As such the design isn't ready for implementation yet. Further iterations needs to be made before the design can be considered complete.

This is also reflected in the fact that the design of the individual modules isn't at the same stage for each module e.g. interfaces has been specified for some modules, but not for the other modules. It hasn't been possible to construct a complete and detailed test specification for every module either.

Although the design is not entirely complete, it is still usable. Two general methods has been used in the design of the MCC. Making the design distributed enables the possibility of executing each module on a separate hardware platform if needed. The use of XML has the advantage of human-readable inter module messages, thereby enabling easy access to debug information and manual control. Since the Message Handler handles incoming XML messages, it is also possible to develop new applications that interact with the MCC. This is possible as long as the new applications uses the interfaces and XML defined by the design of the MCC.

Each module description includes one or more activity diagrams and a class diagram. These diagrams defines a first iteration of which functionality needs to be implemented in each module.

As specified previously further iterations will be needed to complete the design. These iterations should besides, completing the requirement specification, also result in a ICD with traceable interfaces for each data type and method used in the MCC. To complete the design of the MCC, a design of the GSS and the AS must also be present to dertermine the complete GENSO ICD.

11 Conclusion

The GENSO project aims at creating a large scale distributed network of ground stations to facilitate student satellite communication. This network can expand the overall communication with a given satellite making a satellite mission more efficient. The principles of such a network has been implemented an used in practice by other universities and space agencies. However, GENSO wishes to create the network with a new design and additional features.

This project group has chosen to design the Mission Control Client (MCC) node of the GENSO network, the other nodes being a Ground Station Server (GSS) and an Authentication Server (AS). To accomplish this, the group has participated in the structuring of the GENSO requirement specification. Through this a use case analysis of the MCC has been made to create a link between the GENSO objectives and the requirements. Furthermore the GENSO requirements structure has been implemented in a DokuWiki to make a division in subsystems and to make the requirements traceable.

Due to the incomplete definitions of the nodes in the GENSO network, the actual requirements however, are not complete, entirely testable or detailed enough to complete the design at this point. The current result of this project is therefore incomplete with no implementation and test results. The state of the design is what it was possible to achieve with the work effort on the requirement structure an the fact of the project duration of both this project and the GENSO project. The duration of this project is one semester while GENSO has a work schedule of $2\frac{1}{2}$ years.

This project is to be regarded as a first iteration of the MCC in the GENSO project. The work of the project group does not take account of a number of the GENSO requirements as it was chosen to leave it out in the first iteration of the system. It has been accomplished to make a design of the MCC as far as possible according the GENSO requirement specification. It should now be possible for the GENSO project to be inspired and to continue the work of developing a large scale distributed system of ground stations.

12 Perspectives

The perspectives of this project are many, but the main concern would be evaluation of the GENSO project and it's influence of this AAU project. The GENSO project has already from the start been different from ordinary projects at Aalborg University.

In this project the GENSO team has been considered as customer, providing wishes and requirements to the product. This product development has been estimated to $2\frac{1}{2}$ years of work by the GENSO team, which of course gives some problems when an AAU project group tries to implement it all in a one semester. In the start of the semester, the group has expected a fully specified requirements document provided by the GENSO team, but it was early realized that it was necessary to join the requirements work to construct a usable requirements specification. This has of course affected the implementation of the whole project.

A possible alternative to the strategy used in the project could be a more specific project defined by Aalborg University. The project group have used a lot of time defining the project. If the project has been more defined from start, it would properly reach the implementation state.

The recommendation from the project group to the GENSO team, based on experience working with the project will then be:

- Start with small iterations of the system, which makes it possible to implement a base system which is ready for expansion afterwards. This would make it possible to have an overall view of the whole system, implementing features step by step.
- The workpackage division would have to be much later in the project, making everybody work on the requirements and initial design. When the design and features are fixed, the workpakages can be defined by logical design parts with strictly defined interfaces.
- Use more specific dedicated time on requirements, defining what is really needed for the first beta version of the software. This groups experience is that trying to implement everything by waterfall model would not do the job.

If this group were continuing with the project, the next step would probably be to finish design of the MCC by resolving the TBCs and TBDs from the requirements. The will of course have influence on the design section. It would be impossible to develop and implement the system from the requirements that were presented at first. The implementation, of the system specified by a real requirements specification, will be done by using iterative development. This is usually be mentioned as the spiral method. It is the groups conviction that this is the best way implementing such software project as GENSO.

Bibliography

[1] Stephen Biering-Sørensen. *Håndbog i Struktureret Program Udvikling*. Ingeniøren bøger, 1st edition, 2002. ISBN: 87-571-1046-8.

[2] Bo Andersen, Claus Grøn, Rasmus Hviid Knudsen, Claus Nielsen, and Kresten Kjær Sørensen. *Documentation for SSETI-Express and AAUSAT-II Ground station*. Aalborg Universitet, 2004. Project period: September 2nd - December 17th.

[3] CelesTrack. *CelesTrack TLE description*. `http://www.celestrak.com/columns/v04n03/`, 2006. Accessed 20.12.06.

[4] DokuWiki. *DokuWiki*. `http://wiki.splitbrain.org/wiki:dokuwiki`, 2006. Accessed 20.11.06.

[5] Hans-Erik Eriksson. *UML^{TM} 2 Toolkit*. Wiley Publishing Inc., 2004. ISBN: 0-471-46361-2.

[6] ESA. *Types of orbit*. `http://www.esa.int/esaSC/SEMU4QS1VED_index_0.html`, 2006. Accessed 20.12.06.

[7] James Cutler, Peder Linder, and Armando Fox. *A Federated Ground Station Network*. Stanford University, 2002. Space System Development Laboratory.

[8] KD2BD John A. Magliacane. *Predict website*. `http://www.qsl.net/kd2bd/predict.html`, 2006. Accessed 20.12.06.

[9] Kristian Edlund, Martin Nygaard Kragelund, Rasmus Stougaard Nielsen, Axel Gottlieb Michelsen, and Martin Green. *Mission Control Center*. Aalborg Universitet, 2004. Project period: September 2nd - December 17th.

[10] Neil Melville. *GENSO Implementation Plan*. pdf on CD-ROM, 2006.

[11] Neil Melville. *GENSO Objectives and Preliminary Requirements*. pdf on CD-ROM, 2006.

[12] NORAD. *Website*. `http://www.norad.mil`, 2006. Accessed 20.12.06.

[13] Mercury Design Team. *Mercury Ground Station Reference Model – Version 0.2.0*. `http://mgsn.sourceforge.net/docs/mdoelv0.2.0.php`, 2002. Accessed 20.12.06.

[14] James R. Wertz. *Mission Geometry; Orbit and Constellation Design and Management.* Microcosm Press and Kluwer Academic Publishers, 1st edition, 2001. ISBN: 1-881883-07-8.

[15] Wikipedia. *Satellite.* `http://en.wikipedia.org/wiki/Satelite`, 2006. Accessed 20.12.06.

[16] Wikipedia. *XML.* `http://en.wikipedia.org/wiki/XML`, 2006. Accessed 20.12.06.

A Satellite trajectories

Student satellites, and communication satellites in general, are always flying in orbits around the Earth, and these orbits are also known as geocentric orbits[15]. The physical principles for keeping the satellite in orbit are due to:

- the centripetal force
- the satellites high velocity

The centripetal force is due to the gravitational force between the satelites and Earth (see Figure A.1). Due to the Earth's larger mass when compared to the mass of the satellite the force is inward towards the center of Earth as shown on the figure.

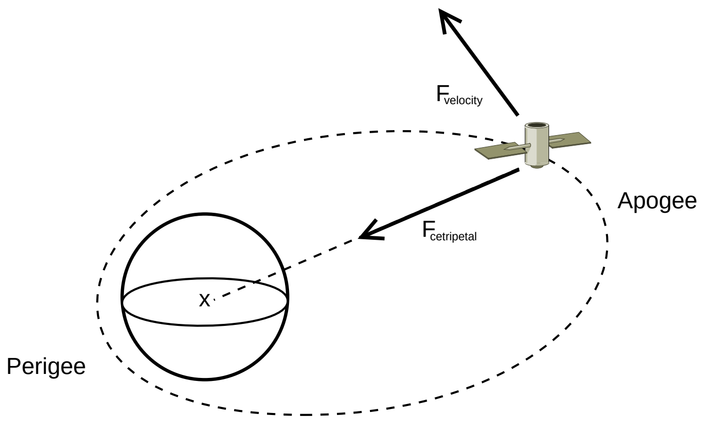


Figure A.1: The figure shows the different forces acting on the satellite and thereby keeping the satellite in its trajectory. The gravitational force causes an inward force, while the satellites velocity causes an outward force.

The satellite travels with a high velocity (several km/s) and thereby causes a outward force to act on the satellite. Once the inward and outward forces acting on the satellite are equal, the satellite will keep it's trajectory, which is illustrated by the dotted kurve on Figure A.1.

The type of orbit, a given satellite is in, can be described by characteristics such as the satellites altitude, the satellites inclination to Earths equator and the direction in which the satellite orbits Earth[6]. Depending on a satellites altitude above Earth it can be classified in the following categories:

- Low Earth orbit (in the range from 0 - 2000km).

- Medium Earth orbit (in the range from 2000 - 35786km).
- High Earth orbit (above 35786km).

Besides the satellites altitude the orbit can also be classified according to it's inclination (see Figure A.2):

- Polar orbit (where the satellites inclination is (or close to) 90 degrees).
- Polar Sun-synchronous orbit (same as polar orbit but the satellite passes equator at the same local time at each pass).

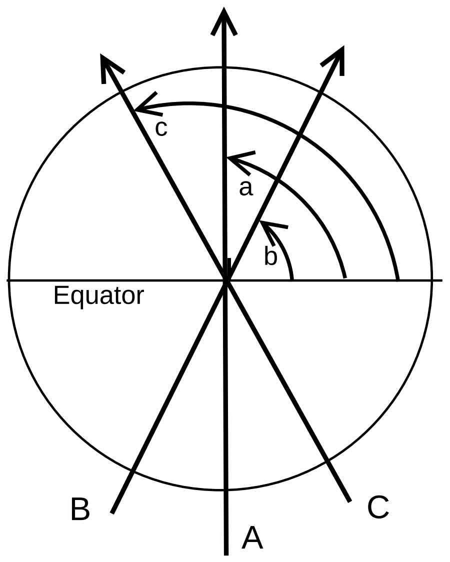


Figure A.2: The figure shows the different types of inclination orbits. A polor orbit is marked by A and the corresponding 90°angle a. B and the angle b shows a prograde orbit with an inclination angle less than 90°. Finally C and the angle c shows a retrograde orbit with an inclination angle above 90°.

If the satellite follows Equator and thereby has an inclination of 0°the orbit isn't inclined. If the inclination is less than 90°the satellites orbit has the same direction as the Earth rotates, also know as a prograde orbit. If the inclination is more than 90°the orbit is called a retrograde orbit.

The type of orbit, a given satellite uses, dependings on what type of applications the satellite is expected to do. If it's a weater satellite that has to monitor the weater in Europe fx., the satellite is put into a geostationary orbit. If it's a student satellite, with a small power supply, the satellite uses a Low Earth Orbit, to lower the use of power for communication between the satellite and the ground station (as well as lowering the cost of launch) .

Once the satellite is in orbit the ground station needs to know when a satellite passes above the ground station. This can be calculated since satellite trajectories are always elliptic. This will be further described below.

Due to the low budgets used on student satellites, the satellites are usually launched as “cargo” on commercial launches. The student satellite is encapsulated in a launch pod and this pod is then ejected from the launch rocket when the rocket reaches the right altitude. Finaly the student satellite is ejected from the pod by a feather mechanism. Therefore it is impossible to predict the satellites actual trajectory pre-launch time.

The North American Aerospace Defense Command(NORAD)[12] observes space and will report observations of newly discovered objects. The observations made by NORAD can then be used to calculate the student satellites trajectory.

Once a satellite has been observed three times in it’s orbit, it is possible to calculate the six Keplerian quantities:

- Semi-major axis
- Eccentricity
- Inclination
- Argument of perigee
- Time of perigee passage
- Celestial longitude of the ascending node

The eccentricity of an ellipse is defined as the distance between the two foci devided by the major axis. The eccentricity is actually a measurement for how much an ellipse deviates from a circle (a circle has a eccentricity of zero.).

Perigee and apogee are also important terms when describing satellite trajectories. Perigee is the term used for the point in a satellites trajectory, where the satellite is closest to Earth. Apogee is used for the point where the satellite is farthest away from Earth.

The terms semi-major axis, eccentricity and inclination has already been explained above. The point on the trajectory where the satellite crosses the planets equatorial plane is called the ascending node if the satellites trajectory goes towards North. The point is called the descending node if the satellite passes towards South (see Figure A.3 on the next page).

The argument of perigee is the argument of the perigee from the ascending node. The time of perigee passage is equal to the time when the satellite passes perigee and last quantity is the ascending nodes longitude.

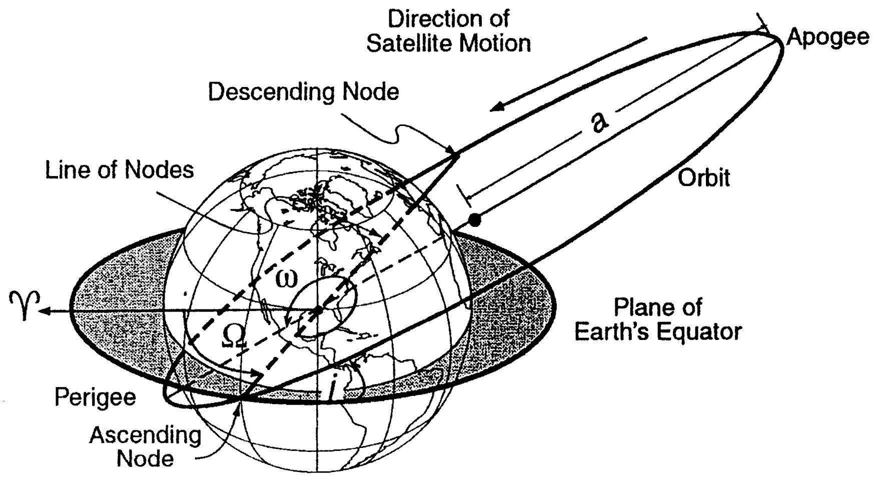


Figure A.3: The figure shows a satellites trajectory around the Earth. The dotted area is the Earths equatorial plane. Υ shows the direction of the vernal equinox. The angle Ω is measured in the equatorial plane, while ω is measured in the orbit plane. a shows the orbits semi major axis and the angle i the inclination. [14, p. 47]

The student satellites from AAU are in orbits classified as Polar Low Earth orbits with an altitude of approximately 850 km and an inclination of approximately 90°.

B Description of NORAD TLE

A TLE is a Two Line Element data set describing the orbital specifications of a given satellite. These two lines consist of 69-characters each (including white spaces). They can be used together with NORAD's orbital models to determine the position at a given time, and the connected velocity of a satellite. All international letters (A-Z and 0-9) are allowed including space, plus and minus signs.

The format can be described on the following form[3]:

```
1 NNNNNC NNNNNAAA NNNNN.NNNNNNNN +.NNNNNNNN +NNNNN-N +NNNNN-N N NNNNN
2 NNNNN NNN.NNNN NNN.NNNN NNNNNNN NNN.NNNN NNN.NNNN NN.NNNNNNNNNNNNNN
```

- N represents a 0-9 digit or some times a space.
- A represents a A-Z character or a space.
- C represents the classification, the public TLE's this will be U for unclassified, but it can be S for secret.
- + represents a plus, minus or space character.
- - represents a plus or minus sign

Each value will put another set of boundaries to the values. The following schemes will give an overview of each column of the lines:

Character:	Description:
01	Line Number of Element Data
03-07	Satellite Number
08	Classification
10-11	International Designator (Last two digits of launch year)
12-14	International Designator (Launch number of the year)
15-17	International Designator (Piece of the launch)
19-20	Epoch Year (Last two digits of year)
21-32	Epoch (Day of the year and fractional portion of the day)
34-43	First Time Derivative of the Mean Motion
45-52	Second Time Derivative of Mean Motion (decimal point assumed)
54-61	BSTAR drag term (decimal point assumed)
63	Ephemeris type
65-68	Element number
69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)

Table B.1: Two-Line Element Set Format Definition, Line 1

Character:	Description:
01	Line Number of Element Data
03-07	Satellite Number
09-16	Inclination [Degrees]
18-25	Right Ascension of the Ascending Node [Degrees]
27-33	Eccentricity (decimal point assumed)
35-42	Argument of Perigee [Degrees]
44-51	Mean Anomaly [Degrees]
53-63	Mean Motion [Revs per day]
64-68	Revolution number at epoch [Revs]
69	Checksum (Modulo 10)

Table B.2: Two-Line Element Set Format Definition, Line 2

C Accepttest specification

This chapter describes how to accomplish the acceptest of the requirement specification in chapter 7 on page 37. The test is split into sequences describing the test of one or more requirements.

Sequence 1

This sequence will test the requirements 1.11 and 1.12 This test requires a list of locations of the ground stations in the network and a result from Predict for each ground station.

Actions:

1. The user enters a satellite ID
2. The user specifies a period below 24 hours.
3. The user selects "Calculate flight plan"

Success criteria:

- The flight plan is shown to the user as a table containing entries of 1.12
- The flight plan is shown to the user as a map containing entries of 1.12
- The ground stations in the flight plan matches the coresponding ground stations in the ground station list.
- The pass time of each ground station matches the result of Predict

Sequence 2

This sequence will test the requirements 1.13 and 1.14

Actions:

1. The user chooses a number of the ground station from the flight plan
2. The user chooses to view capabilities of one of the ground stations

Success criteria:

- The ground stations in the table is marked at the users choice
- The ground stations in the map is marked at the users choice
- The capabilities of 1.14 is shown the the user upon choice

Sequence 3

This sequence will test the requirements 1.21 and 1.22

Actions:

1. The user selects "Book ground stations" to book the selected ground stations

Success criteria:

- It is shown to the user which ground stations have accepted and rejected the booking
- Each booking contains the information from 1.21

Sequence 4

This sequence will test the requirements 1.31 and 1.32. As it is TBD which encryption should be used, the test of these requirements is not specified yet.

Sequence 5

This sequence will test requirement 1.34

Actions:

1. The user sends and receives satellite mission data
2. The user chooses to view the data from the database

Success criteria:

- The data packets are shown to the user identified by the entries in 1.34

Sequence 6

This sequence will test requirement 1.35

Actions:

1. The user sends and receives satellite mission data from one GSS
2. The user request the pass report from the GSS

Success criteria:

- The pass report is shown to the user containing the entries of 1.35

Sequence 7

This sequence will test requirements 1.36 and 1.37

Actions:

1. The MCC is connected to a GSS for the user to communicate with a satellite

Success criteria:

- An indicator in the user interface shows to the user whether the MCC and the GSS is connected or not
- The data packet count for the current session is shown to the user
- The total data packet count is shown to the user

Sequence 8

This sequence will test requirements 1.41 and 1.42

Actions:

1. The MCC is connected to a GSS for the user to communicate with a satellite
2. The user changes a setting on the GSS

Success criteria:

- The current GSS settings are shown to the user

- The GSS settings are changeable
- The effect of the changes are visible in the downlink satellite mission data.

Sequence 9

This sequence will test requirements 1.51, 1.52, 1.53 and 1.54

Actions:

1. The user chooses to change the local MCC settings
2. The user chooses to change the settings of the local space craft

Success criteria:

- The local MCC settings are shown to the user
- It is possible for the user to change the MCC settings
- The local space craft settings are shown to the user with the entries of 1.52
- It is possible for the user to change the space craft settings

Sequence 10

This sequence will test requirements 1.61 and 1.62

Actions:

1. The user wants to logon the MCC
2. The user is done using the MCC and wants to logoff

Success criteria:

- It is possible to input username and password
- It is possible to choose "Logoff"

Sequence 11

This sequence will test requirements 1.71 and 1.72

Actions:

1. The user chooses "Debug mode" to see node and module communication

Success criteria:

- The node and module communication is shown to the user as human readable tekst.

D Acronyms

AAU Aalborg University

AMSAT The Radio Amateur Satellite Corporation

API Application Programming Interface

AS Authentication Server

ASL Authentication Server List

CES Central Server

ESA European Space Agency

FGN Federal Ground Station Network

GENSO Global Educational Network for Satellite Operations

GSMS Ground Station Management Service

GSML Ground Station Markup Language

GSS Ground Station Server

GSSL Ground Station Server List

GUI Graphical user interface

ICD Interface Control Document

IPC Inter Process Communication

ISEB International Space Education Board

JAXA Japan Aerospace Exploration Agency

MCC Mission Control Client

MGSN Mercury Ground Station Network

NASA National Aeronautics and Space Administration

OPS Operation Server

PSL Participating Satellite List

RQT	Requirements
SSETI	Student Space Exploration and Technology Initiative
TBC	To Be Confirmed
TBD	To Be Determined
TLE	Two-Line Element
UI	User Interface
UML	Unified Modeling Language
UNISEC	University Space Engineering Consortium
XML	Extensible Markup Language
RAMS	Reliability, Availability, Maintainability and Safety
CSSI	Center for Space Standards and Innovation
NORAD	North American Aerospace Defense Command

E Glossary of terms

Pass A pass is when a satellite is in the range of a ground station

Satellite mission data Satellite mission data is the data exchanged between the satellite and the satellite operator/application.

Satellite application A satellite application is the application used by the satellite operator to send and receive satellite mission data to/from the satellite.

Communication session A communication session is when a satellite application communicates with the satellite. This is achieved by connecting a satellite application to the MCC which forwards the satellite mission data to a GSS. The GSS will take care of the radio communication with the satellite. A communication session ends when the satellite is out of range.

Flight plan A flight plan is a set of satellite passes which is available for reservation or already reserved.

Time window A time window is the time between a given start time and a given end time.