Title:

Distributed Control Room for the AAU CubeSat

Group:

01gr733

Members of the Group:

Mikael Bo Andersen	Martin Nielsen
Gunnar Sigurðarson	Jørn G. Larsen
Kenneth Sørensen	Svend Møller

Supervisor:

Per Printz Madsen

Project period:

September 2nd – December 19th 2001-12-16

Number of pages: 72

Number printed: 9

Introduction to the worksheets

The students at Aalborg University have started building a small satellite. The design of the satellite follows the CubeSat concept. A control room is needed for the satellite as a part of the ground segment. This project deals with the development of this control room. The control room is distributed to make it possible for the users to control the satellite over the Internet.

The pre-analysis in this document describes what a ground segment. Furthermore it is analysed which protocols and standards are best suited in the construction of a distributed control room. The result of the pre-analysis is that PHP, HTML, HTTP and HTTPS is chosen as the technologies for designing the control room.

In the analysis the system is split into three components: the first is a database, which is the central part of the system. Here data about the satellite is stored. The next component is the communication manager that is the interface between satellite and database. Finally the graphics user interface provides the interface between users and the database.

All of these three components are analysed but only the graphical user interface and the database is designed and implemented.

The design describes the two components in detail and they are extended in order to provide the functionality asked for.. Parts of the two components are implemented as a prototype in order to be able test if the design is correct.

Table of contents

List of figures

List of tables

1 Pre-analysis

1.1 What is a ground segment?

The purpose of this section is to give the reader an overview of what a ground segment for supporting a satellite is, and how others have implemented such system.

First, the general structure will be described, and how users are interacting with the system and how data are represented in the system[1].

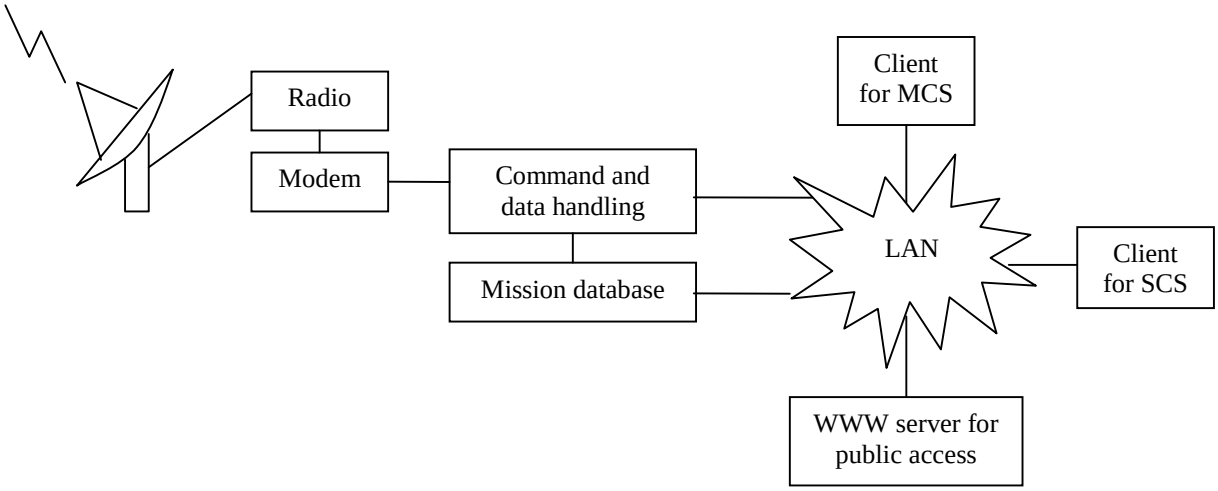


Figure 1.1 Structure of a typical ground segment

[1] **Article: JSWITCH/JSAT: REAL-TIME AND OFFLINE WORLD WIDE WEB INTERFACE**

http://isc.gsfc.nasa.gov/Papers/DOC/JSWITCH.PDF, December 2001

The role of Centralization in a Distributed Architecture: The SCOS II Experience

http://www.esoc.esa.de/external/mso/SpaceOps/2_09/2_09.pdf, December 2001

Radio and modem
The low-level communication between ground segment and satellite is maintained by radio. The radio is connected to the command and data-handling unit by a modem. In order to make the communication reliable a lower layer network protocol is introduced.
Command and data handling
This unit maintains the high-level communication between ground and satellite. Commands are sent and received from the satellite. Before the commands are sent to the satellite, they are verified and scheduled in order to check that command sequences are valid. The data from the satellite is received and distributed to the clients that need them. This unit may also carry out alarm and event handling from the satellite.
Mission database
All commands and data are stored in this database in order to keep track of the events of the satellite.
Client for MCS and SCS
To represent data from the satellite, client software is needed. It can be a large screen to represent all the data from the satellite or a special client for every type of user. The clients can be placed locally and connected by LAN or they can be remote and connected via the Internet or some other kind of remote access.
Server for public access
On some missions, the public has access to view data from the satellite, for example photos. Typically, this access is realised by a web-server, which enables the public to access parts of the mission data through their web-browser.
LAN
All units at the ground segment are connected through a local area network. This may be a standard network like an Ethernet or a real time network depending on the situation.

1.2 Users

The users that make use of a ground segment can be categorized as a satellite, mission control staff (MCS), scientific staff (SCS) and public users. Those users will be described in the section below.

Satellite
The Satellite uses the ground station to inform other users about status and events of the satellite and to send the information that the payload gathers to the users. The communication between the ground station and the satellite has to include some sort of identification to make sure that only one satellite is communicating with the ground station.
Mission control staff
The roles of these users are to control and monitor the satellite. This includes that they have to look at the housekeeping data and act if there is a need for that. They also have to act when the satellite sends alarms. These users are authorized to view and change a part of the parameters in the satellite.
Scientific staff
The scientific users have access to the data that is gathered by the payload of the satellite, this data is often private. Scientific staff has been granted access to view and change parameters related to the mission or payload of the satellite.
Public users
In some missions, the public have limited access to view data from the satellite.

1.3 Data

The data in a typical ground segment for a spacecraft is based on telemetry and tele commands. Telemetry is data transmitted from the spacecraft to the ground. Conversely, tele commands are data send to the spacecraft. Telemetry and Tele commands are represented as small packages, or frames, which are transmitted via a radio link between the spacecraft and the ground station. To keep track of the life of the spacecraft these data are stored in a database.

Telemetry
The onboard telemetry data are created in two categories, housekeeping and scientific data. The sensor readings on board the satellite creates the housekeeping and the payload creates scientific data.
Tele commands
Tele commands are issued by ground personal to control the satellite. It can be commands that tell it what to do or it can be requests for information, which then are transmitted as telemetry from the satellite. The Tele commands has to be coordinated in order to make sure that the satellite is in a state in which the commands are valid.

1.4 The AAU CubeSat ground segment

The structure of the AAU CubeSat ground segment is build up of four modules, showed in Figure 1.2. The "com" module handles the communication to and from the satellite and includes a radio and modem. This module will only be analysed and not designed or implemented in this project. The main area this project will cover is the developing of a server system and the clients that interact with it.

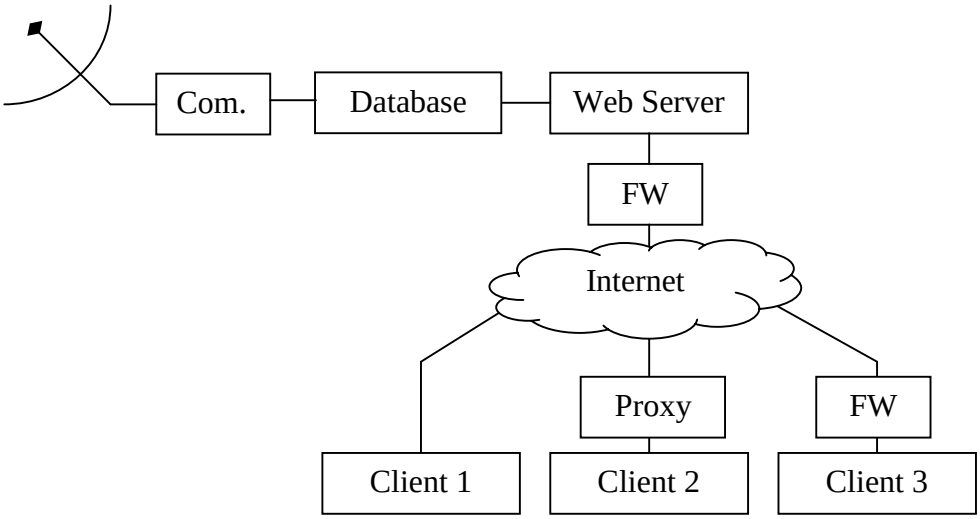


Figure 1.2 CubeSat ground segment

The first topic that has to be covered is how clients and server should communication with each other. What type of protocol should be used and what type of protecting system should the communication work on. Demands, to the communication between client and server are shown in the list below.

- It should be possible to log on to the system from "everywhere" on the Internet
- Some kind of security must exist to protect communication from unauthorized persons
- Communication through proxies and firewalls should be possible
- The system must be platform independent
- An encrypted protocol must be used to secure communication
- Commands sent from a client must be handled within a specified time

In appendix I to VIII the possible protocols are described. From this, it was chosen to use HTTP and HTTPS protocol to realise the communication between clients and server.

1.5 System definition

A distributed control room for the AAU CubeSat has to be constructed. A distribution via the Internet using PHP to generate HTML documents and SSL to encrypt the communication must be implemented. Users should be able to access data stored in a MySQL database at different access levels. A Linux server should contain the database and an Apache web server. The communication with the satellite is analysed but not design, implemented and tested.

Functionality	Communication with the CubeSat Make the students from the sub-systems able to communicate with their sub-systems in the satellite Remote communication via Internet using HTTP and HTTPS protocols
Application Domain	Students from other subsystems 5th semester group with communication Web-access for public users Web-access for staff (students)
Conditions	Used by students at AAU Most users have knowledge about software and programming Poor specifications from the start The software is a service, which is used by other students Must be distributed system
Technology	HTTP / HTTPS PHP MySQL database C++ PC Linux
Object system	Telemetry (housekeeping, log, image, memory dump) Tele commands (memory update, take picture, ???) Images Users
Responsibility	The system is responsible for the distribution of the control room

2 Analysis

In this chapter the analysis of the system is described. First part is an overview of the whole system divided into three components. Each component is analysed independently and described in details.

2.1 Component diagram for the system

The system is divided into three components. The graphics user interface is used to provide access to the system for all the users. The database is used for storing information in the system and to provide a synchronization mechanism between communication manager and the graphics user interface. The communication manager is responsible for the communication with the satellite. The component and their relations are shown in Figure 2.1.

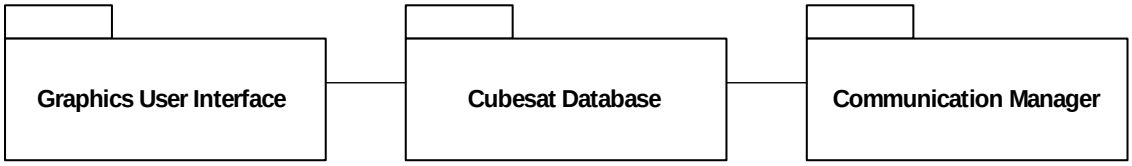


Figure 2.1 Component diagram for the system

Each of these subsystems is described in details in the following chapters.

2.2 Graphics user interface

The graphics user interface (GUI) component acts like a link between the users and the system. This analysis is based on the technology used for this component. It will be implemented as a number of homepages on a website and the functionality is a number of server scripts.

2.2.1 Users

The system has different type of users. Each user was found in the preanalysis and their purposes were described. In table 2.1 each user is described and a new type of user, administrator is introduced.

User type	Description
Administrator	This user has the total control of the system. There is no limitations
Mission control staff	This user is responsible to control the satellite. The only limitation is user administration
Scientific staff	These users can access all the information received from the satellite. They can not perform any changes to the system
Public	These users have only limited access to the system

Table 2.1 List of user and their propose

Each user has access to a number of tasks. Each task is shown in table 2.2.

Task name	Description
Status	Display a short summery of the chosen variables from the system.
Image	Viewer for the images received from the satellite.
Log	Displays the log from the satellite
Housekeeping (HK)	Displays the housekeeping data from the satellite
MemoryManager	Manage the memory in the satellite
Flight plan	Management system for the flight plan. This included adding, editing and deleting of different types of tasks
Administrate	Administration of the users who need to access the system. It will be possible to add, edit and delete users.

Table 2.2 Different types of tasks in the system

The different types of users have only access to some of the tasks. This is depicted in figure 2.2.

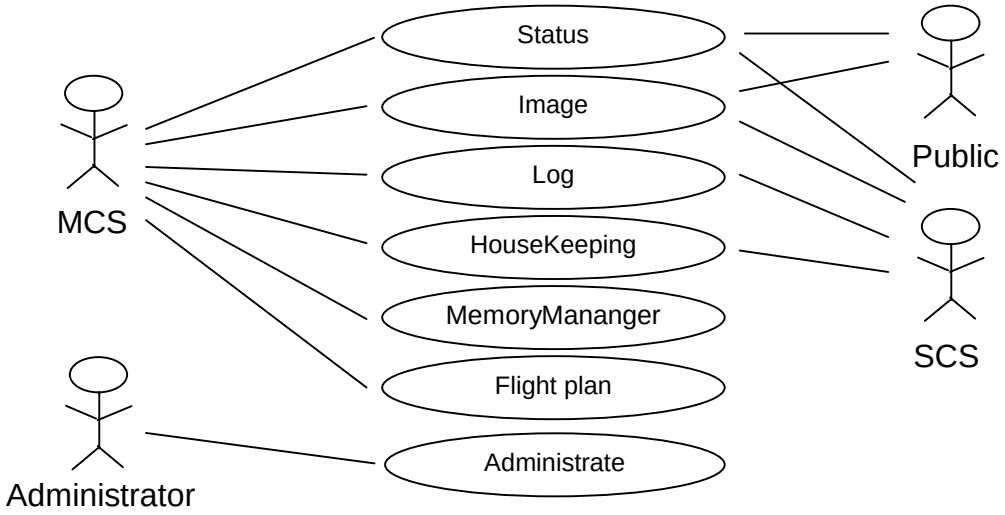


Figure 2.2 User case diagram for the different users and the tasks in the system

2.2.2 Display structure

The homepage is divided into several frames each with a special purpose. The frame division is depicted in figure 2.3.

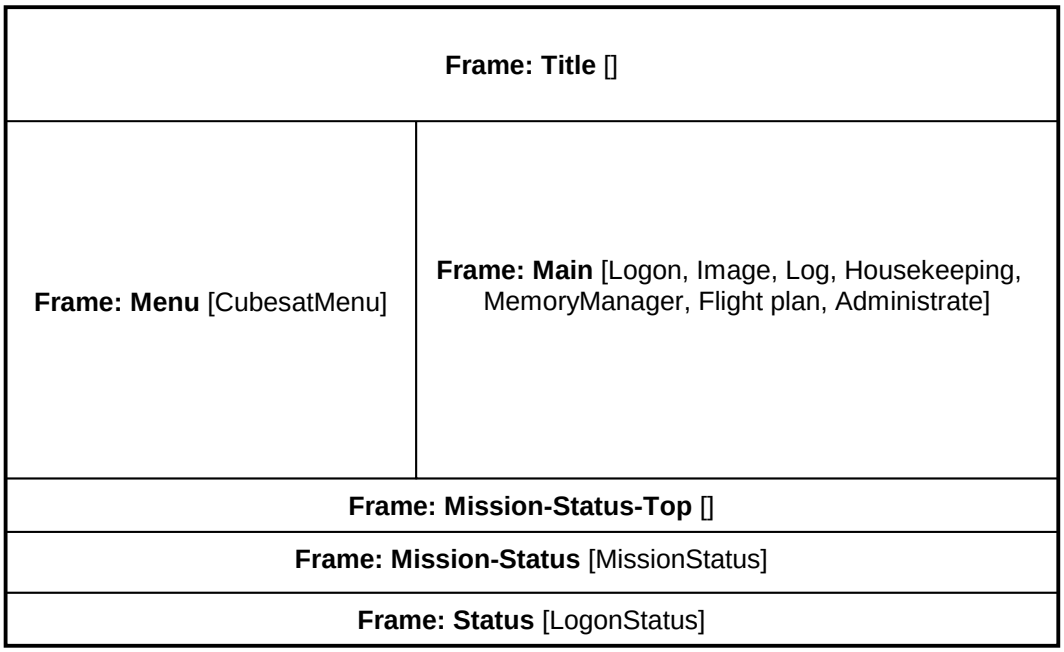


Figure 2.3 Overview of the graphics user interface view

2.2.3 Overview of the system

Combining the system tasks and the display structure, a class diagram is constructed. This provides an overview of the relations in the graphics user interface and is depicted in figure 2.4.

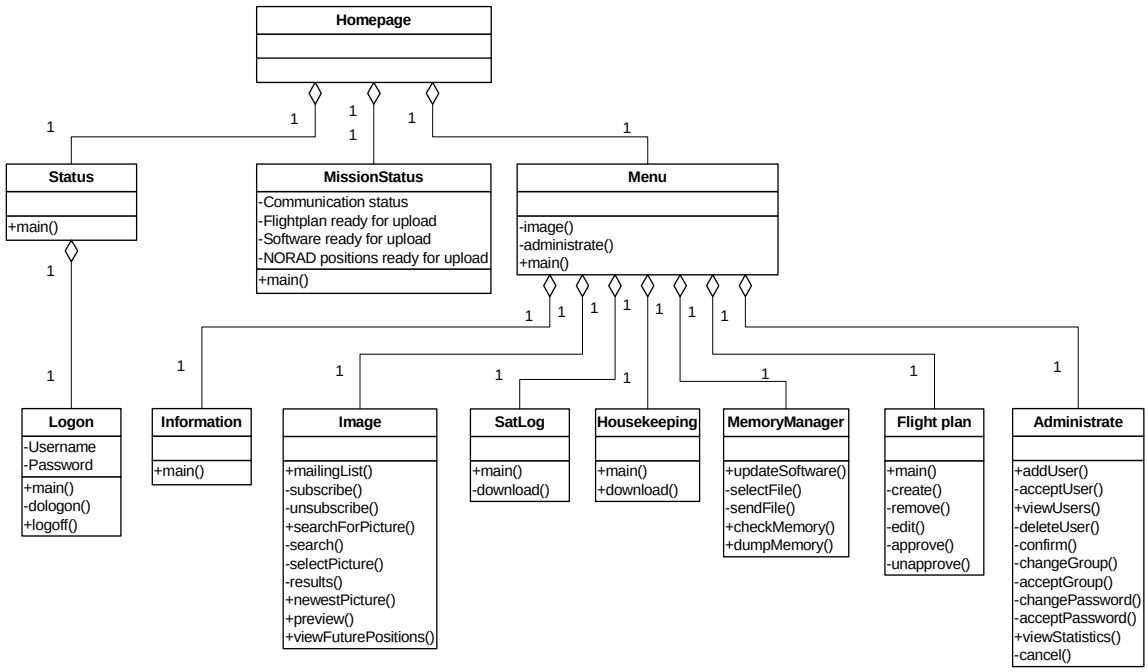


Figure 2.4 The structure of the graphics user interface

The different classes will be described in details in the following sections.

Class Status

This class provides the features to logoff the system. The functions are described in table 2.3.

Name	Complexity	Type	Description
main	”Link”	Read	Display the current authorization of the user

Table 2.3 Function description of the LogonStatus class

Class MissionStatus

Selected information gathered by the satellite is displayed by this class. The only function is update and it is automatically called every 10 seconds. The function is described in table 2.4.

Name	Complexity	Type	Description
main	”Link”	Read	This function displays a list of log entries or housekeeping entries

Table 2.4 Function description of the MissionStatus class

Class Menu

The menu is used to select the different tasks in the system. The functions are described in table 2.5.

Name	Complexity	Type	Description
image	”Link”	Read	Expand the image submenu
administrate	“Link”	Read	Expand the administrate submenu
main	Low	Read	Show the default menu

Table 2.5 Function description of the Menu class

Class Logon

This class provides the functionality to log on to the system. The functions are described in table 2.6.

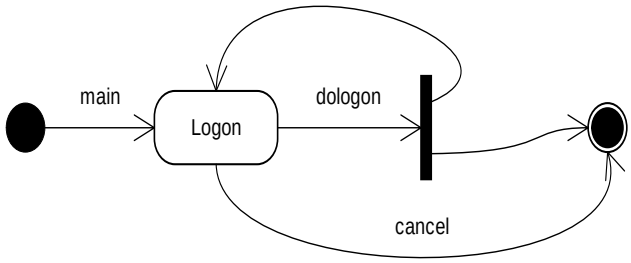


Figure 2.5 Use case for the Logon class

Name	Complexity	Type	Description
main	”Link”	Read	Display the logon form
dologon	“Link”	Update	Performs the actual logon
logoff	“Link”	Read	Log the user off the system

Table 2.6 Function description of the Log class and the Housekeeping class

Class Information

This class shows some default message the all the users. The functions are described in table 2.6.

Name	Complexity	Type	Description
main	”Link”	Read	Display the information form

Table 2.7 Function description of the Log class and the Housekeeping class

Class Image

This class provides functionality to view the images taken from the satellite. It will be possible to select the newest picture, search for a specific one and get a preview of the pictures taken. There will also be some possibility for subscribing to a mailing list and get a mail when a picture successfully transferred to the ground.

The use of these functions is described in the use case depicted in figure 2.6.

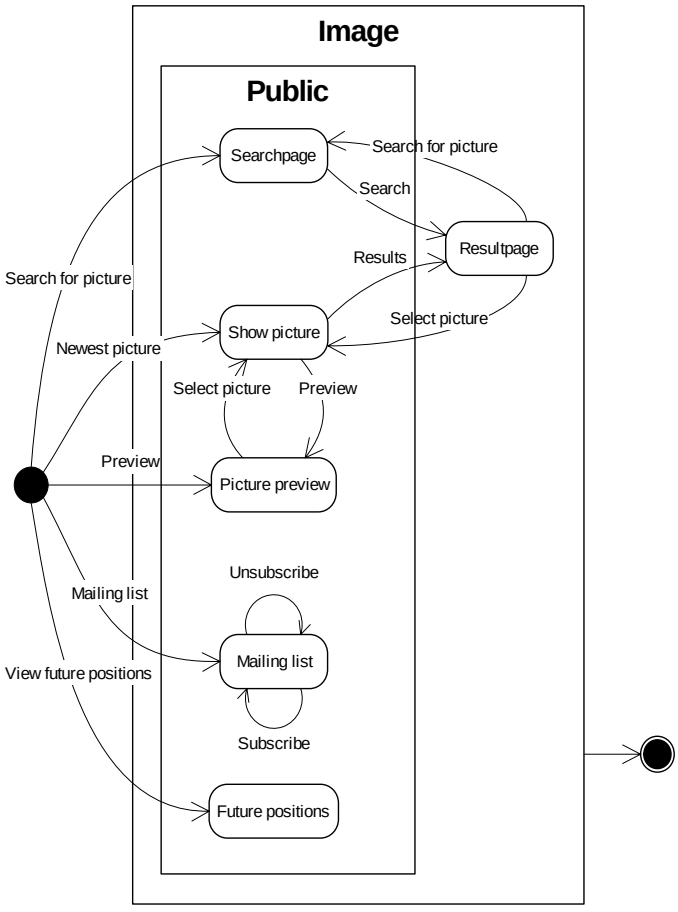


Figure 2.6 Use case for class Image

A list of the function in this class is described in details in table 2.8.

Name	Complexity	Type	Description
mailingList	“Link”		Simple “html” link on the main homepage to the mailing list homepage
searchForPicture	“Link”		Simple “html” link on the main homepage to the search homepage
subscribe	Low	Update	Having entered first name, surname and email address the user can subscribe the mailing list. The email address is written in the database
unsubscribe	Low	Update	Having entered first name, surname and email address the user can unsubscribe the mailing list. The email address is deleted from the database
search	High	Read	Using input specified by the user on the search homepage, the database is searched for pictures relevant to display. The search result is used to make a number of links that will display the picture (select picture)
selectPicture	“Link”		Simple html link on the result homepage and the picture preview homepage to the show picture homepage (made by search or preview)
newestPicture	“Link”		Link made on main homepage
preview	Medium	Read	Display a preview of every picture in the database. The database must be searched to make links to every picture (select picture)
viewFuturePositions	Medium	Read	Display a list of the future positions for the satellite to take a picture. Information is read from the database to display a total list
results	Low		If a search has previously been carried out the result of this will be displayed again. If no search has been carried out no result is displayed

Table 2.8 Function description for the Image class

Class SatLog and class Housekeeping

These classes provide the functionality to access the information gathered by the satellite. There are only two functions described in table 2.9.

Name	Complexity	Type	Description
main	”Link”	Read	This function displays a list of log entries or housekeeping entries
download	“Link”	Read	This function generates a data file with the log or housekeeping data

Table 2.9 Function description of the Log class and the Housekeeping class

Class MemoryManager

This class provides the functionality to manage the memory in the satellite. It will be possible to read, upload and verify the software in the satellite. The functions used for this is described in table 2.10

Name	Complexity	Type	Description
updateSoftware	High	Update	It must be possibly to perform memory updates in the onboard computer.
selectFile	”Link”		Display a from where a software update is placed
sendFile	Low	Update	Stores a file in the system
checkMemory	High	Read	Check part of memory for errors such as bit flips
dumpMemory	High	Read	Dump part of the memory on the onboard computer to earth

Table 2.10 Function description of the MemoryManager class

Class FlightPlan

This class provides the possibility to create, edit and remove tasks in the flight plan. Once a set of tasks is created it can be approved in order to tell the system that it is ready to be uploaded to the satellite. Conversely the set can be unapproved to withdraw that it is ready for upload. This feature is added to ensure that a part of a set of tasks is not uploaded while it is being edited. The use case for the flightPlan is depicted in figure 2.7.

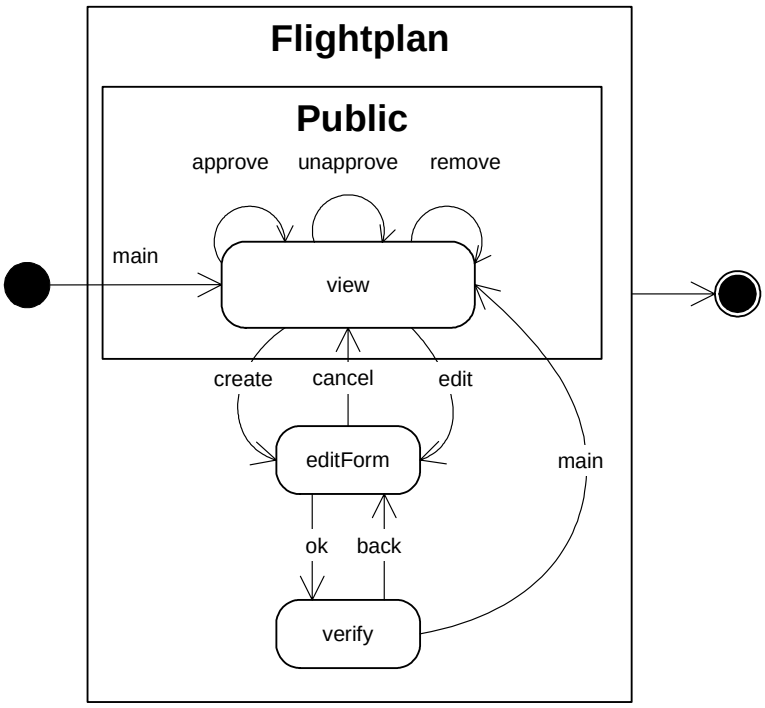


Figure 2.7 Use case for class flightPlan

The functions needed for providing this functionality is described in table 2.11.

Name	Complexity	Type	Description
main	Low	Read	By this flight plan management can be entered. Only one MCS gets access at a time
create	Medium	Update	This function is used to add a task to the flight plan
edit	Medium	Update	This function is used to edit an already existing task
cancel	Low	Update	This function exits the editForm and calls main
ok	Medium	Update	This function verifies user input and stores a task in the database
remove	Low	Update	This function removes a specific task from the flight plan
approve	Low	Update	This function makes the tasks available for upload to the satellite
unapproved	Low	Update	This function makes the tasks unavailable for upload
back	Low	Update	This function is used to return to the editForm if an invalid task is created or edited

Table 2.11 Function description of the FlightPlan class

Class Administrate

User administration is handled by the class Administrate. It provides functionality to view, update, add and delete users. The use of these functions is described in the use case depicted in figure 2.8.

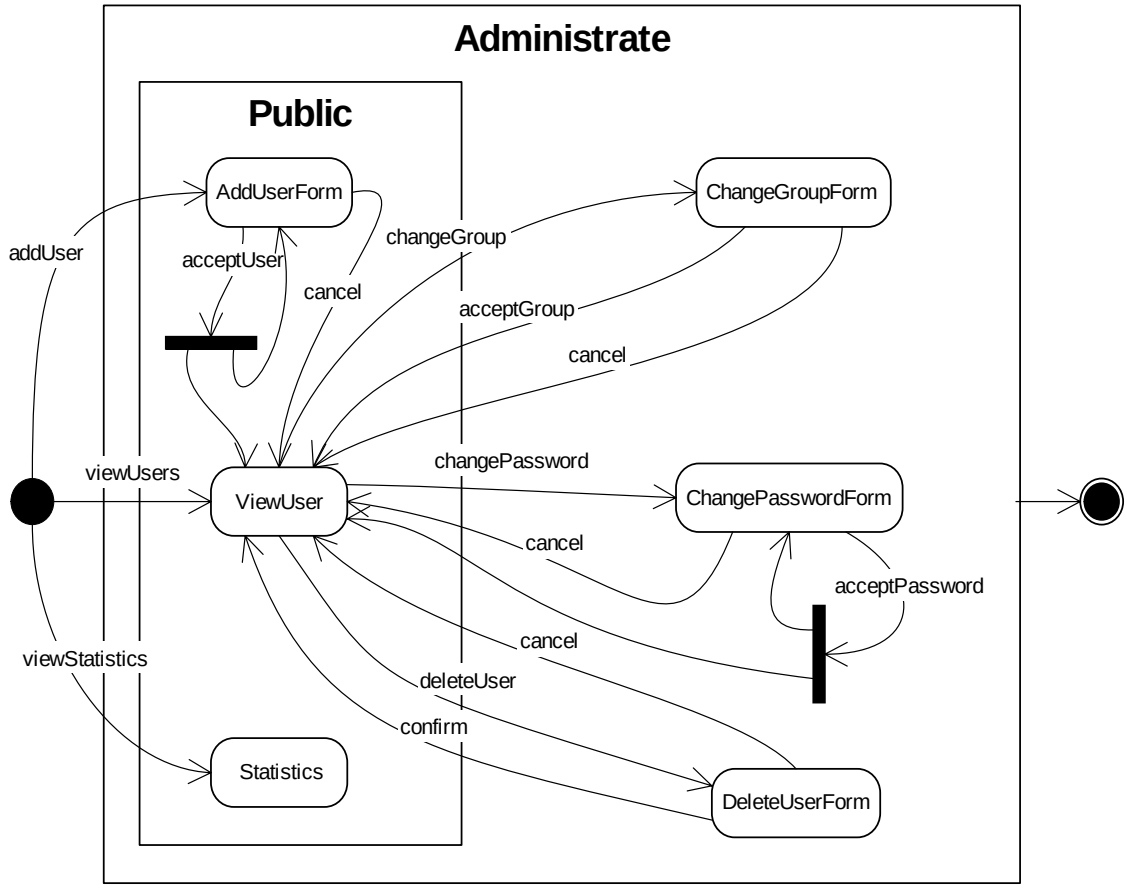


Figure 2.8 Use case for class Administrate

The functions shown in the figure is described in detail in table 2.12.

Name	Complexity	Type	Description
addUser	”Link”		Displays a add user form
acceptUser	Medium	Update	Adds a new user to the system and verifies that no other user with the given username exists
viewUsers	Low	Read	Displays a list of user and the options to delete and change the group or the password of a user
deleteUser	”Link”		Displays a delete user form
confirm	Medium	Update	Delete the user from the system
changeGroup	”Link”		Displays a change group form
acceptGroup	Low	Update	Verifies the new group and updates the user
change password	”Link”		Displays a change password form
acceptPassword	Low	Update	Verifies the new password and updates the user
viewStatistics	Low	Read	Displays the statistics for each user in the system
cancel	Low	Read	Cancels any forms

Table 2.12 Function description of the Administrate class

2.3 CubeSat database

The CubeSat database (CSD) is the core of the system. It defines the way information in the system is stored. It also describes the functionality used to access the stored information. The component is analysed on the assumption that a database is used as the tool to provide the storing of information. All the functions are assumed to be a set of SQL queries.

2.3.1 The tables

The CubeSat database is divided into seven tables. The tables are depicted in figure 2.9 where the tables Log, Housekeeping, FlightPlan and Image are derived from specifications achieved in cooperation with the command and data handling group[2]. The Kepler table is derived from specifications from NORAD[3].

The table Configuration is added to the system to provide a necessary storage for the system state. In addition the table UserControl is added to provide administration of the different users.

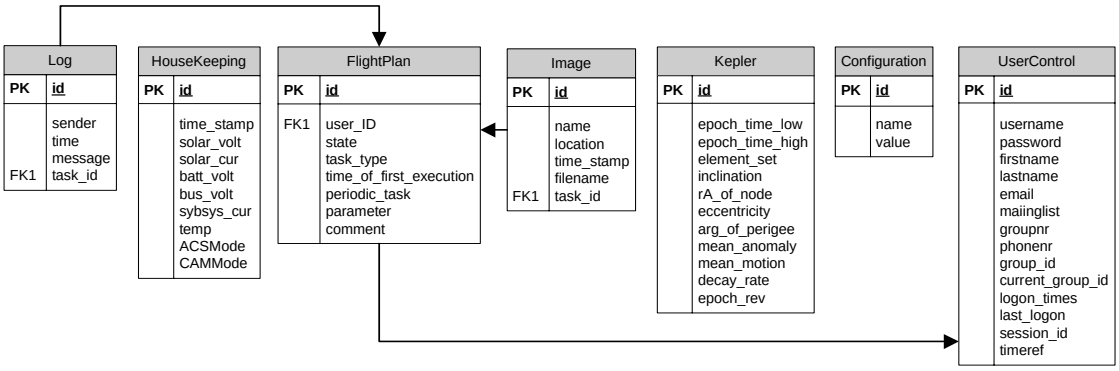


Figure 2.9 Table definition of the cubesat database

Each of these tables has some functions and use cases to describe their behaviour. These are described in the next sections.

Table Kepler

This table is used for storing information about the current position of the satellite. The information is provided by NORAD. The information is updated every second week and will be stored in this table. When the satellite is in range the new position will be send to it. The functions available are described in table 2.14.

Task	Actor	SQL-Type	Description
InsertItem	COMM	insert	Insert a new Kepler element in the table
View	COMM, WEB	select	View the rows in the table and extract the newest Kepler element

[2] http://www.control.auc.dk/~01gr720

[3] http://celestrak.com/NORAD/documentation/tle-fmt.shtml

Table 2.13 Function description for the Kepler table

Table Log

The errors and warnings arising in the satellite is stored in this table to give an overview of the satellite performance and software mishaps. Reading the information can help to detect and correct errors occurring in the satellite. The functions available are described in table 2.14.

Task	Actor	SQL-Type	Description
InsertItem	COMM	insert	Insert a new log element in the table
View	WEB	select	View a set of rows in the table

Table 2.14 Function description for the Log table

Table Housekeeping

The satellite is sampling measurements from the onboard subsystems in given intervals. The information gathered is stored in this table. This will be useful for running overall statistics of the satellites performance. The functions available are described in table 2.15

Task	Actor	SQL-Type	Description
InsertItem	COMM	insert	Insert a new housekeeping element in the table
View	WEB	select	View a set of rows in the table

Table 2.15 Function description for the Housekeeping table

Table FlightPlan

To construct a flight plan for the satellite a number of tasks is put together. Each task has a variable indicating the state of the task. There are five different states and these are described in table 2.16.

State	Description
Created	When created every task enters this state. Now is it possible to change information in the task
Approved	An approved task is ready for transfer to the satellite.
Uploaded	When the communication manager has sent the tasks to the satellite it changes it to this state
Failed	Is set when an execution error is reported by the satellite
Succed	Is set when an successful execution is reported by the satellite
Failed during upload	Is set when the upload of the task failed

Table 2.16 State definition for the Task table

The change of state in a task is done in a specific pattern shown by the state chart in figure 2.10.

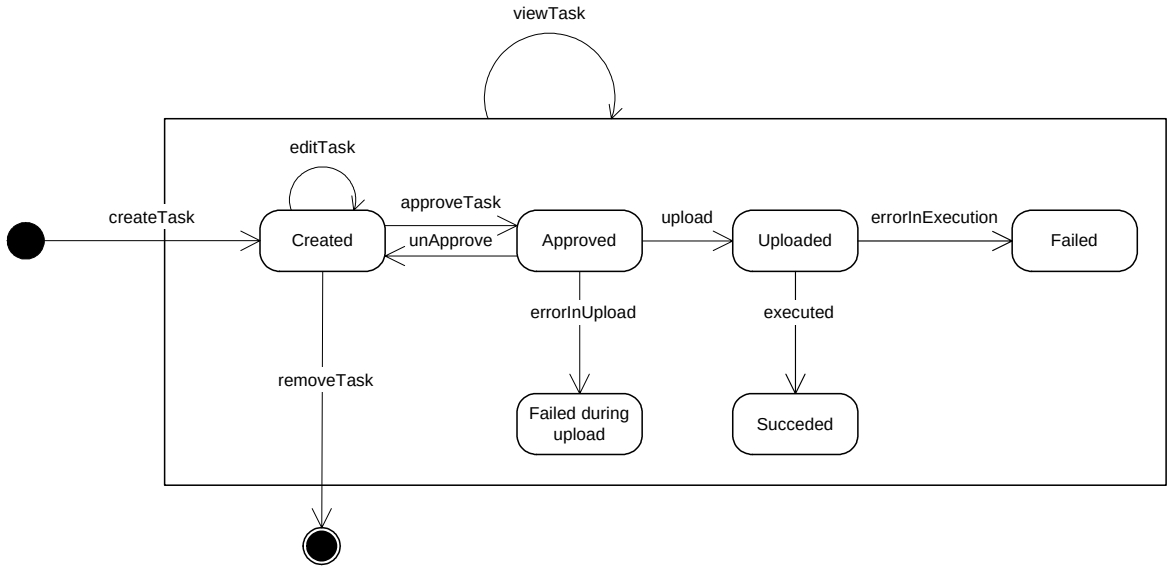


Figure 2.10 State chart for the Task table

The functions needed to change the state are described in table 2.17.

Task	Actor	SQL-Type	Description
createTask	WEB	insert	Insert a new task in the system
editTask	WEB	update	Update a task with new information
removeTask	WEB	delete	Deletes a set of tasks specified by the user
upload	COMM	update	Changes the state of a set of tasks to upload
viewTask	WEB	select	View a set of tasks
approveTasks	WEB	update	Changes the state of a set of tasks to approved
unapproveTasks	WEB	update	Changes the state of a set of tasks to unapproved
errorInUpload	COMM	update	Changes the state of a set of tasks to failed to upload
errorInExecution	COMM	update	Changes the state of a set of tasks to failed
executed	COMM	update	Changes the state of a set of tasks to succeeded

Table 2.17 Function description for the Task table

Table Image

The images taken with the camera in the satellite is send to earth and saved in a file. This table provides information about each of the images taken and a reference to the file. The functions available are described in table 2.18.

Task	Actor	SQL-Type	Description
InsertItem	WEB	insert	Insert information about an upcoming image
Update	COMM	update	Adds a reference to a filename where the image is stored
View	WEB	select	View a set of rows in the table

Table 2.18 Function description for the Image table

Table UserControl

This table stores information about each user for user administration. A user has a variable indicating the state of a user. The states are defined in table 2.19.

State	Description
LoggedOff	The user is not logged into the system
LoggedOn	The user is logged into the system as a specific type

Table 2.19 State definition for the UserControl table

The change of state must be performed in a specific pattern shown in a state chart in figure 2.11.

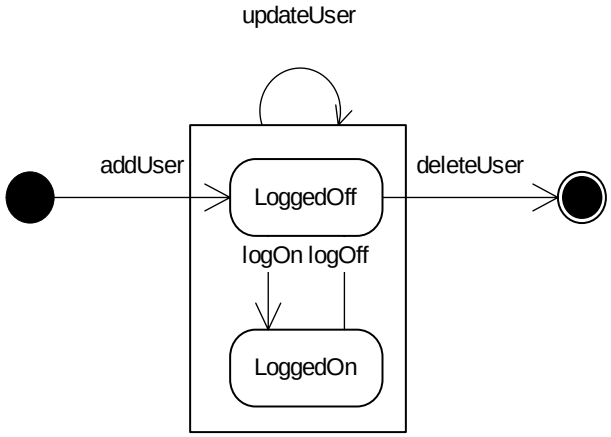


Figure 2.11 State chart for the UserControl class

The function needed to change the state is described in table 2.20.

Task	Actor	SQL-Type	Description
Add	WEB	insert	Insert a new user into the database
Update	WEB	update	Changes information about the user
Delete	WEB	delete	Deletes a user from the system
View	WEB	Select	View a set of users from the table
Logon	WEB	Update	Change the state of a user to logon
Logoff	WEB	update	Change the state of a user to logoff

Table 2.20 Function description for the UserControl table

Table Configuration

This table is used to store the global state of the system. Each entry is specified by a name of the state and the value. The functions available are described in table 2.21.

Task	Actor	SQL-Type	Description
UpdateItem	COMM, WEB	update	Updates a row with a given name with a new value
LockTable	COMM, WEB	lock table	Lock the table for exclusive rights
UnlockTable	COMM, WEB	unlock table	Unlock the table
View	WEB	select	View a set of configuration elements

Table 2.21 Function description for the Configuration table

2.4 Communication manager

The communication manager is the part of the ground station that handles the communication between the database and the spacecraft. The order of the communication is depicted in figure 2.12:

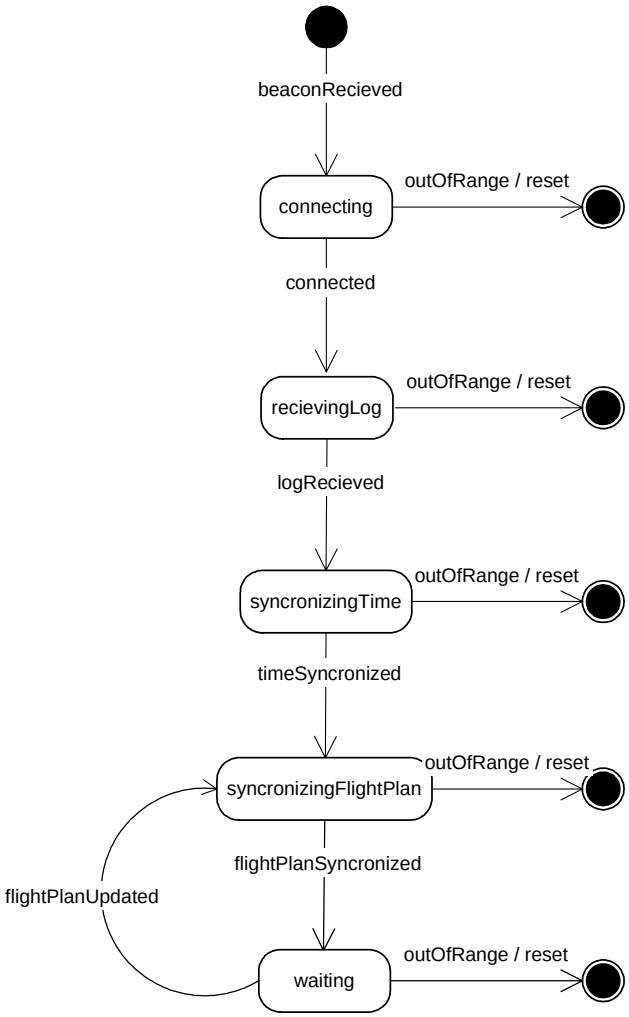


Figure 2.12 Communication between CubeSat and ground station

The communication manager tries to connect when receiving a beacon, if successful; the communication manager waits for the log to arrive. Then a time synchronisation is initiated and next a synchronisation of the flight plan is performed. In the end of the use case, the communication manager waits either for a new flight plan update in the

database or for the satellite to go out of range. In addition to this suspected use case, the satellite can go out of range before the end of the sequence or the mission control staff can reset the satellite main computer.

2.4.1 Class diagram for the communication manager

The communication manager consists of three classes. A class named CubeSatDatabase, a class named SatCom and a class named T55xProtocol. Together these define the model layer for the Communication Manager. The structure between the classes is shown in figure 2.13.

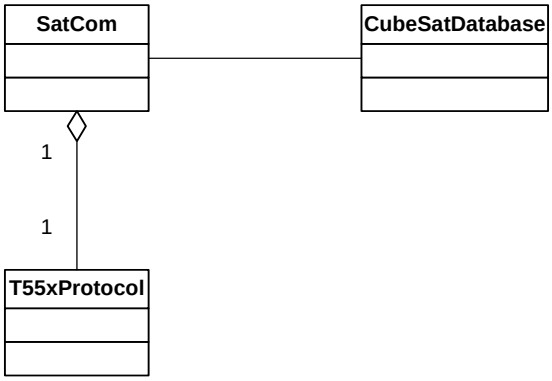


Figure 2.13 Class diagram for the communication manager

The class CubeSatDatabase represents the database for the system; this is associated with the SatCom witch represents the communication part of the satellite. This class aggregates a class T55xProtocol witch handles the actual satellite-ground communication by providing some extra functionality and an encapsulations of the protocol T55x witch others has developed.

2.4.2 State diagrams for the classes in the communication manager

A description of the behaviour of the classes is in the following paragraph, provided by the means of state diagrams.

CubeSatDatabase

The class CubeSatDatabase represents the actual CubeSatDatabase and has one state, and this is the state **running**. When in this state several things can happen. The log can be stored in the database by using putLog, the housekeeping data can be stored in the database by using the putHousekeepingData, and an image can be stored in the database by using putImage. By using uploadFlightPlan, the current flight plan is extracted from the database. ErrorInUploadFlightPlan, errorInExecutionFlightPlan and executedFlightPlan is used to change the state of tasks in the flight plan.

When new keplerdata is received from NORAD the keplerReceived event is triggered, and finally by using getResetStatus the satellite knows if a reset has been requested. This behaviour is depicted in figure 2.14:

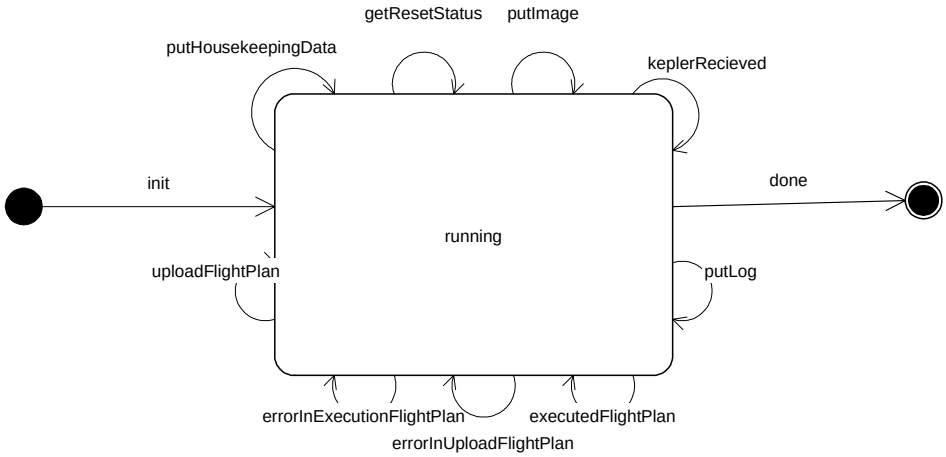


Figure 2.14 State diagram for CubeSatDatabase

SatCom

This class has two states, named outOfRange and inRange. When resuming or pausing the communication with the satellite, the transition between these states happens. See figure 2.15:

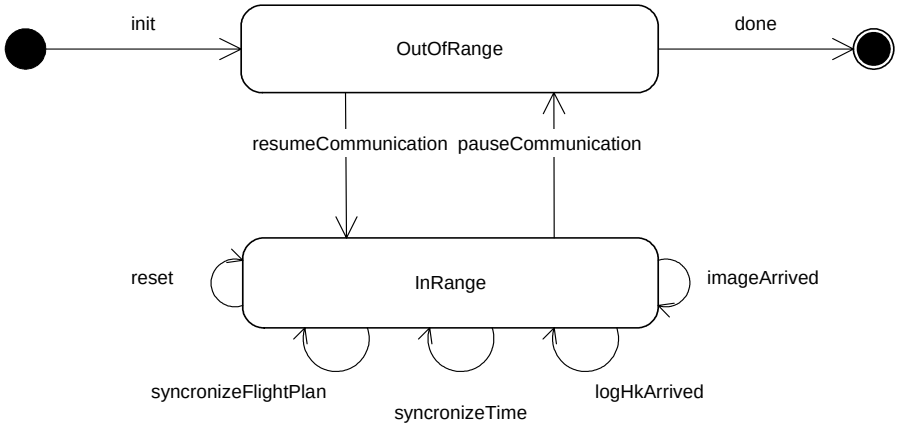


Figure 2.15 State diagram for SatCom

When it is in the state inRange, the following things could happen. The CubeSat database can reset it, synchronize the flight plan or synchronize the time. In addition, an image, log or house keeping data can arrive to the ground segment.

T55xProtocol

This can be in three different states, connectionPaused, when the satellite is not in range, resumeConnection when the satellite is in range but communication has still not been resumed, and finally connectionResumed, which is when the communication between ground segment and the satellite is successfully resumed. The states and transitions between them are depicted in figure 2.16:

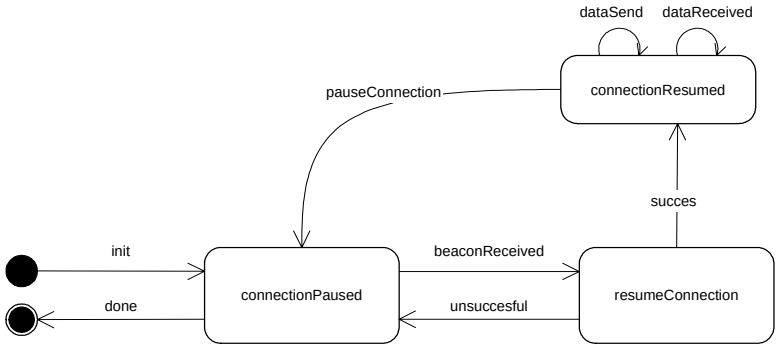


Figure 2.16 State diagram for T55xProtocol

Until receiving a beacon from the satellite, the communication will be paused. At this time, the T55xProtocol will enter the state resumeConnection. In this state, it tries to resume the connection, if this fails, it returns to the former state until receiving a beacon. If the connection is resumed, it enters the state connectionResumed. Three things can happen in this state, either it can send data or data can be received, when these happen, the T55xProtocol remains in the same state.

If the connection is paused by the T55xProtocol, because the satellite is out of range or because the connection fails by other reasons, or because of an activation of a reset, then it enters the state connectionPaused.

2.4.3 Functions

From the state diagrams and the class diagrams following functions have been found.

Class SatCom

The imageArrived function handles an image and sends it to the database. This includes a conversion from the satellite image format to a standard Internet format.

Function synchronizeTime calculates the difference between the satellite and earth time. The difference is added to the earth time, so that the satellite never has to change its time.

Function synchronizeFlightPlan uploads the newest flight plan from the database. By looking at the housekeeping and log information from the satellite, it will be possible to determine which part of the flight plan to be uploaded to the satellite.

Functions name	Complexity	Type
resumeConnection	Medium	Signal
logHKArrived	Low	Update
pauseConnection	Simple	Update
imageArrived	Medium	Update
synchronizeTime	High	Signal
Reset	Low	Update
synchronizeFlightPlan	High	Signal

Table 2.22 Function description for the class SatCom

Class T55XProtocol

The beaconReceived event will take place if the transport protocol has been paused, and after that have received a beacon from the satellite. The beaconRecevied function should be a call back function from T55X API interface and will initiate a resume call to the API.

DataSend and dataReceived functions control the data flow through the T55X protocol. dataReceived should be registered as a call back functions in the T55X API interface.

Functions name	Complexity	Type
beaconReceived	Medium	Signal
dataSend	Medium	Update
pauseConnection	low	Signal
dataReceived	Medium	Update

Table 2.23 Function description for the class T55XProtocol

3 Design

Designing the system consists of two different phases. First phase consist of an overview of the system architecture. Second phase is a detailed description of functionality of the different classes in the system.

Designing the architecture is developing a system structure and specifying design criteria. These affect the design, implementation and test of the software. A criterion is defined as a quality measurement that focuses on one aspect of the design. Therefore is it important to careful select the main criteria.

Specifications of the overall results are important before choosing the design criteria. Therefore, the system is divided into two objectives. The primary objective is the graphics user interface. The cubesat database and communication component are secondary objectives. The design criteria are chosen to concern the primary objective only.

3.1 Primary objective design criteria

The choice of criteria is shown table 3.1 where the importance of each criterion is defined. The rest of the design will focus mainly on fulfilling the most important criteria.

Criteria name	Important	Description
System access	Very high	Insuring the right access to the system is very important. There are two different situations included in this definition. • Users have to fulfil the specified use cases. Thereby the system must be designed in a way that easily verifies the correct use. • Discarding unauthorized users from the system. These users could be hackers trying to grant access the satellite control. Both situations have great impact when designing and testing the software. The difficulty lies in the definition of unwanted use of the system. The derived use cases from the analysis specify the correct use. All other use of the system should be reported. The reporting feature is critical to design because storing the right information is essential in order to correct the problem.

Criteria name	Important	Description
Input validation	Very High	Human interaction requires validation of the input. Ensuring the input correctness could minimize errors in the system There are six aspects in this validation process. • Discarding any unwanted data from the input. It could be keys enter, escape, backspace etc. • Checking for the presence of specific characters. It could be the symbol '@' and '.' in an email address. • Limitations of string lengths are useful when dealing with strings as usernames, passwords and control variables. Both the minimum and the maximum number of characters should be checked. • Non-visible characters. Protecting passwords is essential to maintain a high level of security. This requires an input system, which do not, displays the input from the keyboard. • Type checking of the input. Converting the input to the right type requires some functionality, which is mostly provided by the programming language. An error can occur when converting the input and this must be handled. A conversion could be strings to numbers or strings to dates. Letters that cannot be converted must be handled. • Interval specification. Setting ranges for the numbers or dates must be specified and verified. Fulfilling this requires specific validation patterns that have to be checked. All inputs from the users have to be checked. Another aspect is to report an error when user input cause an execution failure. A correct error reporting can help deriving better validation pattern in future updates of the software.
Testability	High	Test of software is essential but a changeling discipline. Test is used to validate the correctness of the software. There are two aspects of testing. • Test during software implementation. Two different type of test exists. White box testing where all the functionality is tested independently and the black box test used to test the overall functionality of the system. These tests takes a lot of time to specify and perform, but combining design and test considerations can help minimize the time needed to test the software. A problem with these tests is the programmer often gives the perfect input and seldom scenarios occur where the input is wrong. The problem lies in the programmer knowledge of the future users' use of the system. • Test during normal software operation. These tests are more critical because the use of the system is more or less unknown. The real problem in this scenario in the users' use of the system. There are always ways of using the system as programmers never thought about. A good design could help detecting errors and thereby help the programmer to correct the errors. Considerations in the design regarding the testing of the system can help minimize the time needed to test the software and provide easier error detection.

Criteria name	Important	Description
Functionality	Middle	Functionality is the task of designing, implementing and testing separate functions or the total flow in the system. There are different aspects, which are defined below. • Class functionality where each function is carefully defined and verified. This requires a black box test of each function where the input and expected result is defined. This test should verify the correctness of the result of the function given a specific input. Each function can be designed as flowcharts, timeline schematics etc. • System functionality where the flow in the system is important to describe and verify. This is difficult because a system can be designed as a number of processes running local or remote. Here the interaction between the processes is essential but difficult to describe because the flow is nondeterministic. Scenario description can help designing and testing the system. A careful design can help verify the correctness of the system functionality.

Table 3.1 Design criteria for the graphics user interface

Given the design criteria, it is possible to begin the considerations of the system architecture. These are described in the next section.

3.2 Web page design

Because the object-oriented analysis and design method is not developed for web pages, it must be considered how the analysis is used throughout the design. The main results in the analysis were a class diagram, use case diagrams and a number of function descriptions.

The class diagram describes the structure of the system. As many object-oriented programming languages use one file per class, it would be preferable to implement the web page using the same division. This should be no problem when using PHP because one file can include others and thereby gain access to the code of the other classes.

Concerning variables associated to a class they will be implemented as variables in the PHP scripts where needed.

Use case diagrams achieved in the analysis show the intended use of the final program. These along with the function descriptions should be used later on when going into details about the way each function is designed.

3.3 Fulfilling design criteria

The overall design must be considered carefully to fulfil the design criteria. The secure access and input validation criteria require three things to be fulfilled by the design: use case validation, user validation and input validation. The following three sections describe which impact these validations cause on the design.

3.3.1 Use case validation

This type of validation is carried out to ensure that the program cannot end up in a state not considered by the programmer. Looking at the use case diagrams in the analysis it is already specified which transitions are allowed. However, in some way the program has to be able to check which transitions it is carrying out. This could be done by indicating the current state by a variable and then checking if a change to a new state is allowed before executing the function causing the transition. If this validation is implemented in every class, it could easily become a hard time administrating the valid state transitions as it is distributed into a number of files. Due to the common validation for every class, it would be reasonable to create a universal class that handles every use case validation. An advantage in this approach is that information about valid state transitions is collected in one class and therefore easier to administrate.

3.3.2 User validation

From the class diagram, it is clear that some classes must be accessible for some user groups only while others both contain functions available to everybody and functions available to specific user groups only. The universal class handling use case validation is chosen to take care of user validation too. A database is used to store information about functions' availability to user groups.

Validating users is only a poor security if any information about usernames or passwords is intercept by unauthorized persons on the Internet. Thus it would be preferable to implement part of the control room using a secure communication method.

3.3.3 Input validation

The third criterion weighted very high is input validation. Any user input must be checked to be sure that illegal or even damaging information is not entered. The problem must be taken care of by the individual scripts. An input validation class will contain functions for the validations.

3.4 Overall web page structure

The overall structure consist of two elements: Group definition and class diagram of the system

3.4.1 Groups

In the analysis four groups where found. The organization of the group is a hierarchical structure and depicted in figure 3.1.

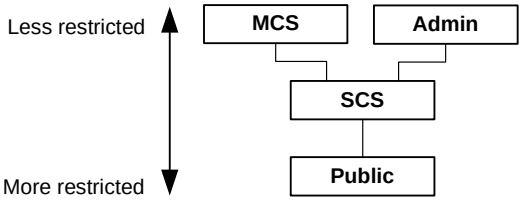


Figure 3.1 Structure of the group organization

The structure divides the groups into three levels, where a higher level means less restrictions. A unique number identifies each group.

Number	Group
1	Admin
2	Mission Control Staff
3	Scientific Staff
4	Public

3.4.2 Class diagram

Based on the analysis and the previous discussion an overall class diagram of the system is designed and the result is shown in figure 3.2.

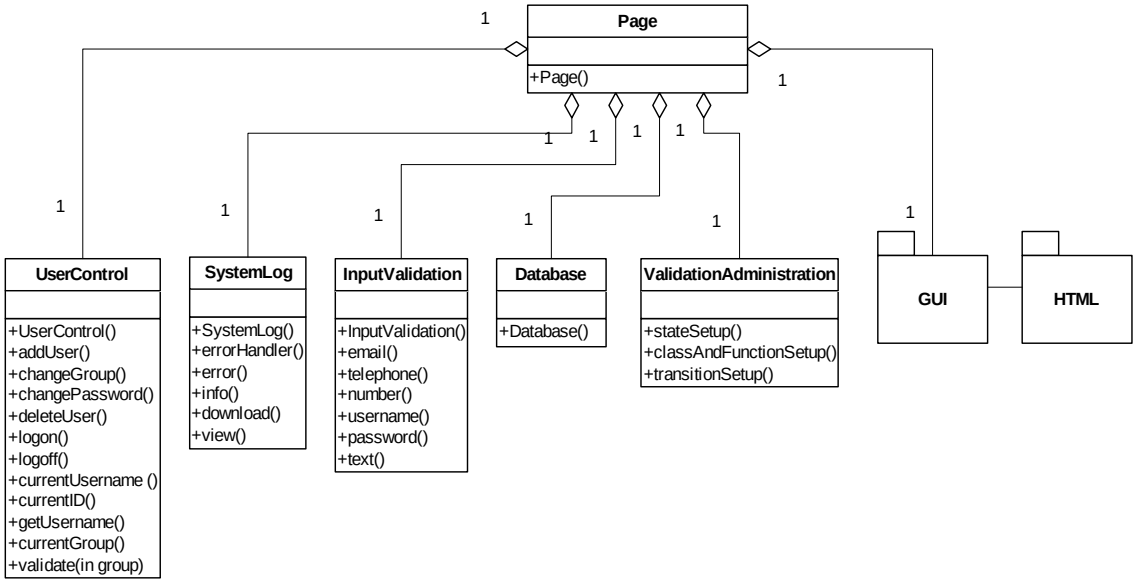


Figure 3.2 Class diagram of the entire system

The system is designed with a central class Page responsible for executing all functions in the system. This class is called every time a user wants to perform anything in the system. All of the criteria found in the previous section in controlled from this class. This eases maintenance of the system and the validation.

There are five classes associated with this class. First the UserControl class provides the necessary functionality to control the user validations. This includes logon and logoff procedures along with validation and administration facilities. Second the SystemLog class provides functions to collect and store errors and information in the system. The InputValidation class provides a selection of different methods to validate user input to the system. The ValidationAdministration class contains functionality to administrate the valid use of the system. Finally, the Database class creates the link to the database.

In addition, two packages where designed the package GUI is derived from the analyses and the HTML is designed to provide an easier way to generate the HTML documents.

3.4.3 Test considerations

Before designing the system, some consideration about the way of testing the system must be performed. There are two important aspects in the test phase:

- Reconstruction of scenarios, which is used to generate errors. This is valuable in both the testing phase and in the final edition.
- Track the user interaction with the system. This is valuable when error reported is cause be a specific way of using the system.

In order to perform these tests enough information must be stored in the database. The problem with this method is the amount of information, which will be stored in the database. To solve this problem some facility to present the information is needed to ease the error correction task.

3.5 The classes of the system

In the next subsection, each of the classes is described with a complete function description and the flow of the complex functions is illustrated with flow charts.

3.5.1 Class Page

The class is responsible for validation of secure communication, users and use cases. The class provides the necessary checks to achieve some of the design criteria.

In order to execute any task in the system this class is called. The way to execute a given functionality is specified by a class variable and a function variable. An example of executing the addUser function in the administrate class is shown below:

```
http://cubesat/page.php?class=administrate&function=addUser&var=mixed
```

The field indicates any variables needed in order to execute the functions. Examples of such could be:

```
subfunction=showForm
username=Cubesat&password=********
```

If the class and function variable is not specified the system should report an error.

Page flow

The flow in this class is basically build as described in figure 3.3 and table 3.2.

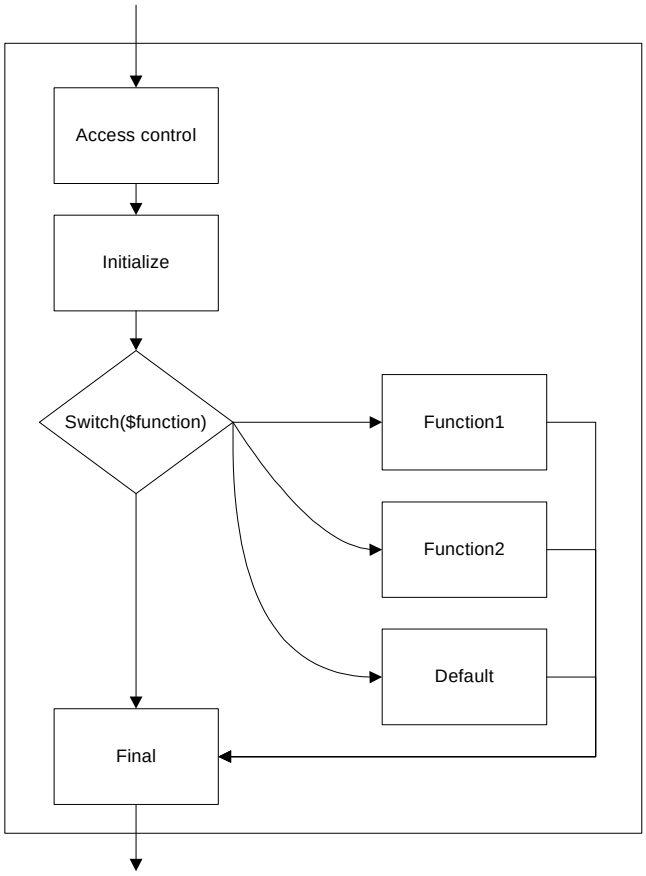


Figure 3.3 Flow diagram of a default script

Task name	Description
Access control	Validates the user, checks the need for a secure communication and checks the state transition
Initialize	Initialization of the class is needed to create an object and set of some variables.
Switch	A variable named function is received from the user and it indicates the function to be run. A switch statement is used to decode the variable and call the appropriate function in the class.
Functions	The decoded function is executed and a result is returned.
Default	If the decoded function does not exist, a default functionality will be called that informs the user about the error.
Final	The final contains functions that are used to end the script, e.g. html end commands and error handling.

Table 3.2 Description of tasks in a default scripts

The different tasks are described in details in the next sections.

Access control

The access controlling contains the three validation tests. Upon successfully execution of the tests, the script continues its flow. The three test systems are described in the next sections.

Secure communication validation

The system to validate the secure communication is depicted in figure 3.4.

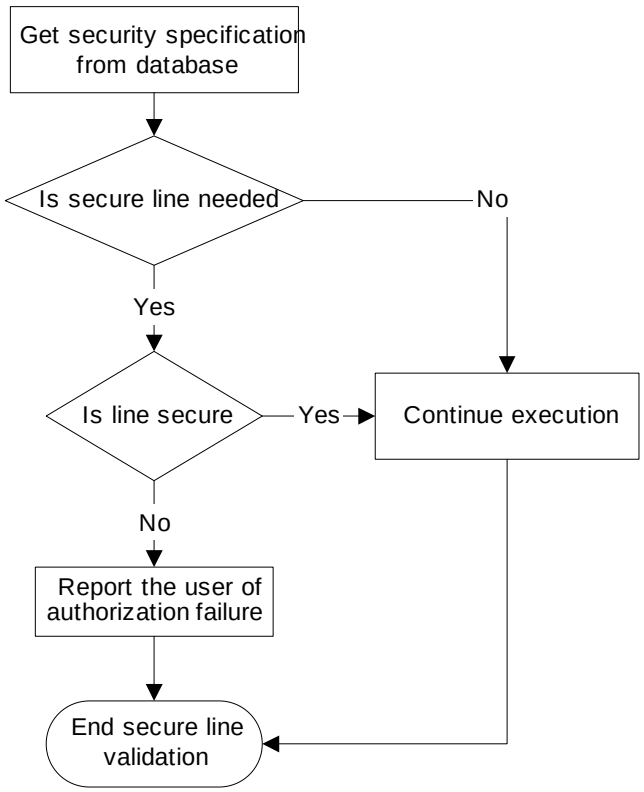


Figure 3.4 Flow diagram of secure communication validation

User validation

The flow to validate a user is shown in figure 3.5.

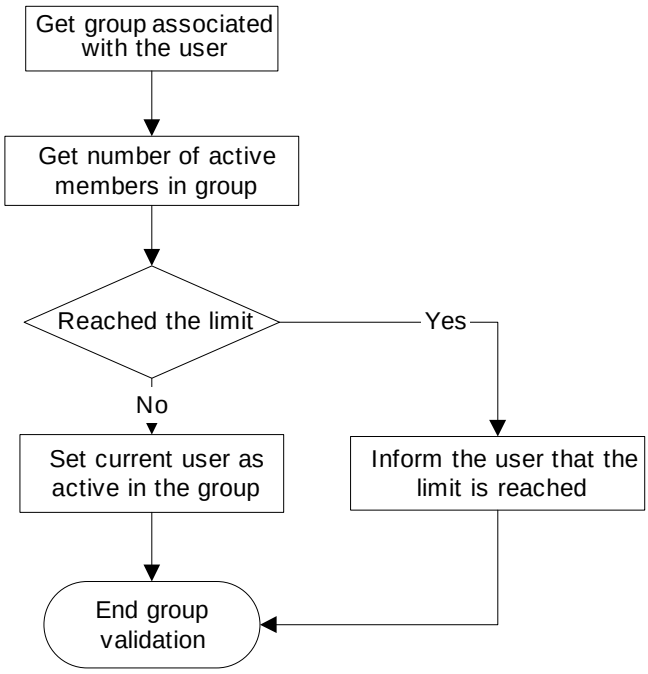


Figure 3.5 Flow diagram of the user validation

Use case validation

The flow to validate the use case and execute the function requested is depicted in figure 3.6.

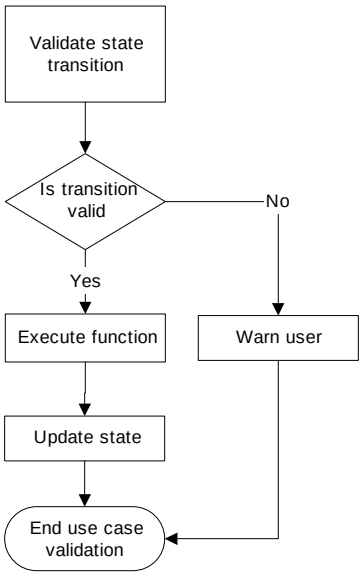


Figure 3.6 Flow diagram of the use case validation

Initialize, switch, function and finalize

The initialize section instantiates the requested class and prepare it for execution. The functions switch, function and default are implemented as part of PHP. Thereby the implementation is easier. The execution flow in depicted in figure 3.7.

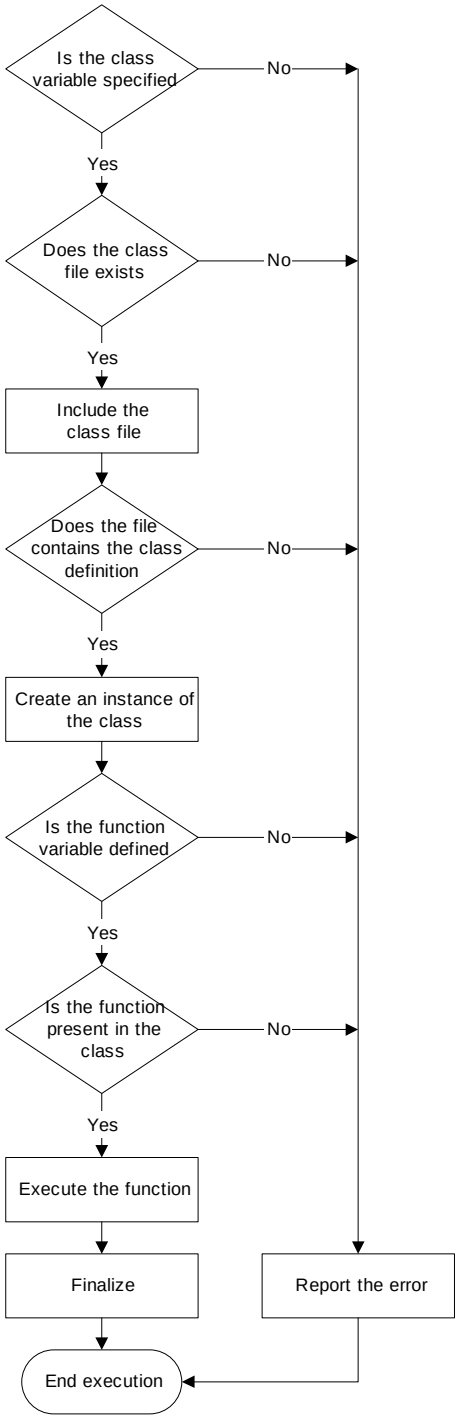


Figure 3.7 Execution of the initialization, functionality and finalization sections

Validation table

The three security checks need information about what is allowed. This information is stored in one table in the database. table 3.3 describes the seven columns necessary to store the information asked for. It is chosen to store the information in a database to make the Page script general. In this manner, no information about the rest of the scripts is stored in the page script.

Fieldname	Type	Special	Nullable	Description
id	int	Primary key	No	Row identification
class	varchar(64)		No	The class containing the function to be called.
function	varchar(64)		No	The name of the function to be called.
currentstate	varchar(64)		No	The state from which it is allowed to call the function specified by class and function.
newstate	varchar(64)		No	The state obtained after execution of the function.
ssl	Enum('Y', 'N')		No	Indicates if a secure line is needed for the function to be executed.
groupID	tinyint		No	Indicates which group the user must be to execute the function specified by class and function.

Table 3.3 Security validation table description

The way the table is created means that a function in a class may be represented in more than one entry. This is due to the fact that some functions may be called from different states and as the state column only specifies one state more entries are needed. table 3.4 shows a case where this is needed.

class	function	currentstate	newstate	ssl	groupID
Image	selectPicture	PreviewPicture	ShowPicture	False	4
Image	selectPicture	Resultpage	ShowPicture	False	4

Table 3.4 Example of security validation table

Another aspect of concern when repeated function entries are possible is inconsistence of the SSL and groupID column. If a function is represented more than once the SSL and groupID data must be exactly the same for every entry. This problem must be taken care of by the administrator of the table. The validation administration tool avoids this problem. However it is possible to insert data manually in the database resulting in inconsistency.

3.5.2 Class Input validation

The input validation class contains functions that validate input. This means checking string lengths, confirming presence or absence of certain characters in a string or verification of valid date input etc. Such auxiliary functions are developed throughout the implementation. The specific functions are not described.

3.5.3 Class UserControl

This class controls all the user validation and identification. The class uses the user_control table in the database to store the information of each user. The specific table design in described in table 3.5

Fieldname	Type	Special	Nullable	Description
Id	int	Primary key	No	Row identification
Username	char(32)		No	The username of the user
Password	char(32)		No	The password of the user
Firstname	char(50)		No	First name of the user
Lastname	char(50)		No	Last name of the user
Email	char(50)		No	Email of the user
Mailinglist	char(34)		No	Specifies if the user wants the information mail
Phonenr	char(50)		No	The phonenumber of the user
Session_id	char(32)		Yes	The current browser session identification. The field is indicating the logon status. An empty field means logged out
Timeref	Bigint		Yes	The field is a timestamp of the last activity
GroupId	Int		No	Group which the user is member of
CurrentGroupId	Int		Yes	The group the user is logged into. This field only differs from the groupId if a second MCS tries to log onto the system. Then the user is logged on as SCS.
LogonTimes	int		No	Records the number of times the user has performed successful a logon
LastLogon	bigint		No	A timestamp indication the last time the user has logged onto the system.

Table 3.5 User control table description

Functions in this class

All functionality is available anywhere.

void addUser()	
Inputs	username – The wanted username password – The wanted password group – The group attached to the user firstname – The first name of the user lastname – The last name of the user email – The email attached to the user telephone – The telephone of the user
Complexity	Low
Category	Update
Trigged / called by	System
Description	Add a new user to the database

void changeGroup()	
Inputs	id – The database identification of the user group – The group to change to
Complexity	Low
Category	Update
Trigged / called by	System
Description	Changes the group of the specified user

void changePassword()	
Inputs	id – The database identification of the user password – The new password oldpassword – The old password of the user
Complexity	Low
Category	Update
Trigged / called by	System
Description	Changes the password of a given user

void deleteUser()	
Inputs	id – The database identification of the user
Complexity	Low
Category	Update
Trigged / called by	System
Description	Delete the specified user from the database

string logon()	
Inputs	username – the name of the user logging on password – the password given by the user
Complexity	Middle
Category	Update
Trigged / called by	System
Description	Logges on a user to the system and ensures only one MCS to log on. A flow chart of this function is depicted in figure 3.8

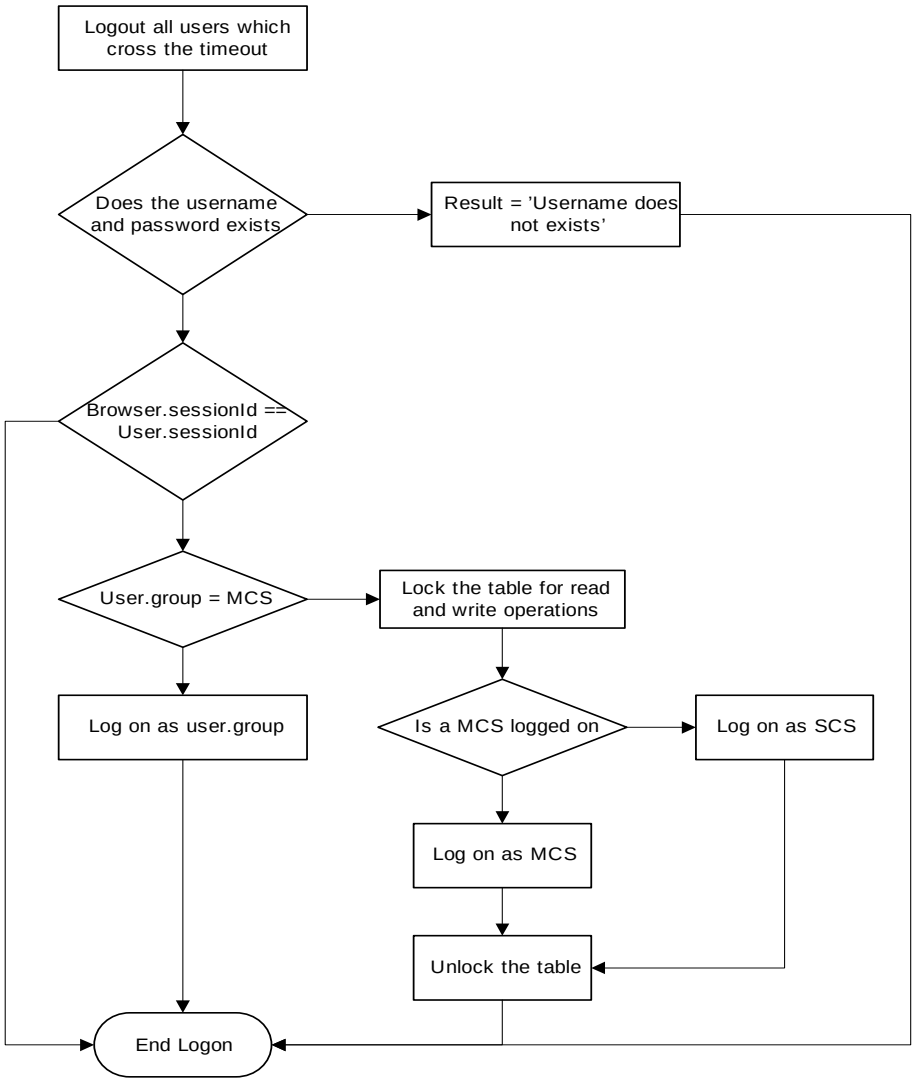


Figure 3.8 Flowchart of the logon function

void logoff()	
Inputs	None
Complexity	Low
Category	Update
Trigged / called by	System
Description	Loges the user out of the system

string currentUsername()	
Inputs	group – the name of the user logging on
Complexity	Low
Category	Read
Trigged / called by	System
Description	Return the username of the current user

int currentID()	
Inputs	None
Complexity	Low
Category	Read
Trigged / called by	System
Description	Return the database identification of the current user

string getUsername()	
Inputs	Id – The database identification of the user
Complexity	Low
Category	Update
Trigged / called by	System
Description	Return the username identified by the provided id

int currentGroup()	
Inputs	None
Complexity	Low
Category	Update
Trigged / called by	System
Description	Returns the group associated with the current user

int validate()	
Inputs	group – Specifies the group which the user must be a member of
Complexity	Middle
Category	Update
Trigged / called by	System
Description	Validate the group of the current user with the group necessary to execute the functionality

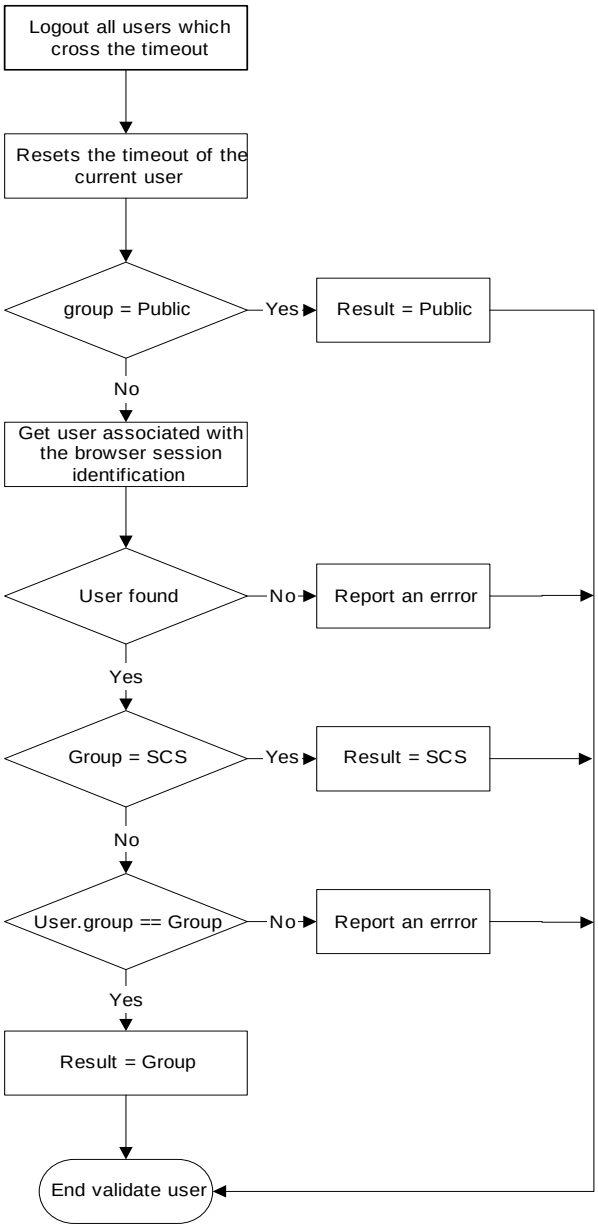


Figure 3.9 Flowchart of the validate function

3.5.4 Class Systemlog

This class provides the functionality to store generated errors in the system. The basic idea with this class is to store enough information in the database and the present it to the developer which is trying to correct the errors. table 3.6 shows the definition of the table used to store the information.

Fieldname	Type	Special	Nullable	Description
id	bigint	Primary key	No	Row identification
dateAndTime	datetime		No	Date and time
currentState	varchar(40)		No	The state variable
linenumber	Int		No	Line number of error occurrence
sourcefile	varchar(255)		No	Source file of the script generating the error
parameters	text		Yes	Parameters send to the script
remoteIP	varchar(15)		No	Ip address of the remote client
sessionID	varchar(32)		No	The identification of the browser
messageType	enum('Info', 'Warning', 'Error')		No	The type of the message
browserType	varchar(60)		No	The type of the browser sending the request
secureLine	enum('Y', 'N')		No	Current secure line status
userId	bigint	user_control.id	No	Reference to the user_control table
function	varchar(30)		No	The function executed
class	varchar(30)		No	The class executed
message	Text		No	User defined message

Table 3.6 Definition of the Syslog table

Functions

The functions in the syslog class are described in this section.

void error()	
Inputs	condition – when this variable is true an error in function call exist line – The line in the file that error exists file – The name of the file that is broken error – Error message sent to the user.
Complexity	Simple
Category	Update
Trigged / called by	system
Description	If an error originates this function is called and the function calls the info function that writes an error entry in the syslog database

void info()	
Inputs	Line – The Line in the file to be logged File – The file name that is logged Message – The message to be written to database MessageType – The type of the message to store (Info)
Complexity	Complex
Category	Update
Trigged / called by	System
Description	When the system is in debug mode or if an error exists the info function writes to syslog database. It also writes information to the screen when an error exist

void view()	
Inputs	None
Complexity	Complex
Category	Read
Trigged / called by	User
Description	This function reads information from the database and writes them to the screen

void download()	
Inputs	None
Complexity	Simple
Category	Read
Trigged / called by	User
Description	With use of this function it is possible to download a log file

3.5.5 Class Validation Administration

This class is an administrative tool used to manage the use case tables that has to be implemented. To make the administration more manageable the tool is divided into three parts. The division means that functions, states and transitions are handled in separate windows and database tables.

Table 3.7 through table 3.9 depicts the tables used in the validation administration. Table 3.7 and table 3.8 are simple and contain both a primary key and a name field to represent states and classes in the system. The “class” table represents a number of tables dynamically created during use of the administration tool. These tables will be named after the classes added to the class table. The fields in the table specify the name of functions in a class, if a secure line is needed to call the function and what group level the user must logged in as to execute the function. Finally the validation table described in table 3.3 is used to store transitions.

Fieldname	Type	Special	Nullable	Description
Sid	Int	Primary key	No	State identification
name	Varchar(64)		No	Name of a state

Table 3.7 States table

Fieldname	Type	Special	Nullable	Description
Cid	Int	Primary key	No	Class identification
Name	Varchar(64)		No	Name of a class

Table 3.8 Classes table

Fieldname	Type	Special	Nullable	Description
Fid	Int	Primary key	No	Function identification
functionname	Varchar(64)		No	Name of a function
SSL	Emun('Y', 'N')		No	Specifies if a secure line in needed
GroupID	Tinyint		No	Specifies the group access level

Table 3.9 "class" table

The following function description goes into details about the three parts of the administration tool.

Class and function setup

void classesAndFunctionSetup()	
Inputs	None
Complexity	Complex
Category	Read
Trigged / called by	User
Description	This function generates the GUI that enables the user to add and delete classes and functions. Furthermore it generates a table containing classes and their functions

void addClass()	
Inputs	classStr – the name of the class to add
Complexity	Simple
Category	Update
Trigged / called by	User
Description	This function is called when the users wants to add a new class. The function adds a class to the class table and creates a new table named as the class itself

void deleteClass()	
Inputs	classChoice – specifies the class to be deleted
Complexity	Medium
Category	Update
Trigged / called by	User
Description	The class specified is deleted from the class table, the table containing the class' functions is deleted and every state transition associated with the class is deleted from the validation table

void addFunction()	
Inputs	functionStr – the name of the function functionclasschoice – the class to add the function to sslneeded – specifies is a secure line is needed to execute the function groupIDchoice – specifies the group access level
Complexity	Simple
Category	Update
Trigged / called by	User
Description	A function is added the table named by the class containing the function. This includes information about the necessity of secure line and group access level

void deleteFunction()	
Inputs	functionChoice – specifies the function to be deleted
Complexity	Medium
Category	Update
Trigged / called by	User
Description	The function specified is deleted from the database. Any transition in the validation table associated to the function is also deleted

State setup

void stateSetup()	
Inputs	None
Complexity	Medium
Category	Read
Trigged / called by	User
Description	This function generates the GUI that enables the user to add and delete states. Furthermore it generates a table containing every state entered

void addState()	
Inputs	stateStr – the name of the state classChoice – the class associated to the state
Complexity	Simple
Category	Update
Trigged / called by	User
Description	The function adds a state to the state table

void deleteState()	
Inputs	stateChoice – specifies the state to be deleted
Complexity	Medium
Category	Update
Trigged / called by	User
Description	The state specified is deleted from the state table and every state transition associated with the state is deleted from the validation table

Transition setup

void transitionSetup()	
Inputs	None
Complexity	Complex
Category	Read
Trigged / called by	User
Description	This function generates the GUI that enables the user to add and delete transitions. Furthermore it generates a table containing all transitions in the database

void addTransition()	
Inputs	classChoice – the class containing the function that results in the transition functionChoice – the function resulting in the transition currentstateChoice – The required state before the function call newstateChoice – the state obtained after the function call
Complexity	Simple
Category	Update
Trigged / called by	User
Description	The function adds a transition to the validation table by the means of the information given in the parameters

void deleteTransition ()	
Inputs	transitionChoice – specifies the transition to be deleted
Complexity	Simple
Category	Update
Trigged / called by	User
Description	The transition specified is deleted from the validation table

3.5.6 GUI Package

This package consist of all the classes found in the section ‘2.2.3 Overview of the system’ in the analysis. Only the classes Image, Flightplan, SatLog and Administrate are described in the next section

Class Image

This class provide all the functionality to gain access the images gather by the satellite.

void Image()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	All
Description	The constructor, setup the default variables and create the default look for the image class

void create()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	Administrator
Description	Creates the database table where images will be save

void mailingList()	
Input	email – The email address button – subscribe and unsubscribe
Complexity	Medium
Category	Read
Trigged / called by	Public, Administrator, MCS and SCS
Description	This function will send an email where it is possibly to reply this email to subscribe or unsubscribe the mailing list

void newestPicture()	
Input	None
Complexity	Simple
Category	Read
Trigged / called by	Public, Administrator, MCS and SCS
Description	This function views the newest picture from the database

void preview()	
Input	id – picture ID for largest picture view
Complexity	Simple
Category	Read
Trigged / called by	Public, Administrator, MCS and SCS
Description	Gives a full view of all pictures in the database and make it possibly for the user to look at one picture in full size

void searchForPicture()	
Input	searchType – Search specification search – Search string
Complexity	Medium
Category	Read
Trigged / called by	Public, Administrator, MCS and SCS
Description	It is possibly to search the database for specific picture

void subscribe()	
Input	code – Identification of the user
Complexity	Medium
Category	Update
Trigged / called by	Public, Administrator, MCS and SCS
Description	When replying to the subscribe email this function will be called. The user are identify by the code variable

void unsubscribe()	
Input	code – Identification of the user
Complexity	Medium
Category	Update
Trigged / called by	Public, Administrator, MCS and SCS
Description	When replying to the unsubscribe email this function will be called. The user are identify by the code variable

void viewFuturePositions()	
Input	None
Complexity	Medium
Category	Read
Trigged / called by	Public, Administrator, MCS and SCS
Description	This function will make it possibly to view future position where a picture will be taking

Class SatLog

The SatLog is the log for the satellite. This class presents log information filtered by patterns set up by the user.

The class SatLog is designed to use a single table in the database. This table is defined as follows:

Fieldname	Type	Special	Nullable	Description
Id	int	Primary key	No	A unique id
timestamp	datetime		No	The time it happened
logType	enum (‘debug’, ‘error’,’status’)		No	The type of log entry
systemName	enum(‘ACS’,’CAM’, ’COM’,’OBC’,’PWS’)		No	The name of the system

The class satLog has three functions. A public function named satLog, a public function named main and a private function named printLog that presents information to the user.

void satLog()	
Input	None
Complexity	Simple
Category	Constructor
Trigged / called by	page.php
Description	This function is a kind of constructor for the satLog class, and it generates header information for the satLog page in the cubesat system

void main()	
Input	None
Complexity	Medium
Category	Read
Trigged / called by	User
Description	This function generates the GUI for the satLog. It also extracts data from the database according to some criteria's specified by the user. The result is sent to printLog

void printLog()	
Input	result – The result from the SQL-query in the database logType – What type of logs there is searched for systemName - What type of system there is searched for sortField – What field the output should be sorted by sortOrder – What order it should be sorted in intevalBegin – The start of an eventual time inteval intevalEnd - The end of an eventual time inteval choice – Contain the selected choice – should the result be shown on screen or saved in a text file
Complexity	Simple
Category	Update
Trigged / called by	satLog->main()
Description	This function present log information on the screen from results provided by satLog->main()

Class FlightPlan

This class is the part of the system, which is used to manage the flight plan. It was derived in the analysis on page 20. The fields of the flightPlan table are described in table 3.10.

Fieldname	Type	Special	Nullable	Description
Id	mediumint	Primary key	No	Task identification
userID	int		No	Id of the user who has created or edited the task
state	enum('created', 'approved', 'uploaded', 'failed', 'succed', 'faileduplоad')		No	The current state of the task
taskType	enum('getStatus', 'takePicture', 'setACSMode')		No	The type of the task
timeOfFirstExecution	Datetime		No	Time of first execution of the task
periodicTask	int unsigned		Yes	Time between executions, if 0 non-periodic task
parameters	Mediumblob		Yes	Parameters to the task
comment	varchar(255)		Yes	The users comment to the task

Table 3.10 flightPlan table

In the following tables the functions of the class flightPlan is described:

void create()	
Inputs	None
Complexity	Simple
Category	Update
Trigged / called by	User
Description	This function is called when the user wants to create a new task. The function parses the control to the function editForm

void edit()	
Inputs	id – the id of the task which is to be edited
Complexity	Simple
Category	Update
Trigged / called by	User
Description	This function is called when the user wants to edit an existing task. The function tests if the task is in the state ‘created’. If this is the case the function editForm is called. Else, nothing happens

void remove()	
Inputs	id – the id of the task which is to be edited
Complexity	Simple
Category	Update
Trigged / called by	User
Description	This function is called when the user wants to remove a task from the flight plan. The function tests whether the task is in the state ‘created’. If this is the case then the task is deleted. Else, nothing happens

void approve()	
Inputs	None
Complexity	Simple
Category	Update
Trigged / called by	User
Description	This function is called when the user wants to approve the flight plan. The function updates all tasks, which are in the state ‘created’ and changes their state to ‘approved’

void unapprove()	
Inputs	None
Complexity	Simple
Category	Update
Trigged / called by	User
Description	This function is called when the user wants to unapprove the flight plan. The function updates all tasks, which are in the state ‘approved’ and changes their state to ‘created’

void editForm()	
Inputs	id: The id of the task that is to be edited. If NULL a new task is created
Complexity	Medium
Category	Read
Trigged / called by	Function edit, function create
Description	This function is called when the user wants to edit or create a task and it generates a form for specifying a task. When a task is to be created the function is called with no parameters and an empty form is displayed. If a task id is specified an existing task is extracted from the database and the data is displayed in the form. The layout of the form will depend on the selected task type such that the input fields will reflect the actual task type

void ok()	
Inputs	id – the id of the task which is to be edited taskType – the selected task type timeOfFirstExecution – time of first execution of the task periodicTask – time between executions, if 0 non-periodic task parameter – parameters to the task comment – the users comment to the task
Complexity	Medium
Category	Update
Trigged / called by	User
Description	This function is called when the user presses ok in the editForm. The function verifies the input from the editForm and stores it in the database

Class Administrate

This class provide all the functionality to administrate the users of the system.

void addUser()	
Input	None
Complexity	Simple
Category	Read
Trigged / called by	All
Description	Display a form with the necessary input in order to create a user. Upon failed input validation the form is displayed with the current valid input from the user

void acceptUser()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	All
Description	Validate the input and adds the new user to the database. Upon failed validation the function addUser is called

void viewUser()	
Input	None
Complexity	Simple
Category	Read
Trigged / called by	All
Description	Displays the current users in the system together with options to delete, change password and change group

void deleteUser()	
Input	None
Complexity	Simple
Category	Read
Trigged / called by	All
Description	Display a form to confirm deletion of a specific user

void confirm()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	All
Description	Perform the deletion of the specified user

void changeGroup()	
Input	None
Complexity	Simple
Category	Read
Trigged / called by	All
Description	Displays a form where a new group for the selected user can be chosen

void acceptGroup()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	All
Description	Changes the group of the selected user

void changePassword()	
Input	None
Complexity	Simple
Category	Read
Trigged / called by	All
Description	Displays a form where the old and new password of the selected user can be entered

void acceptPassword()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	All
Description	Changes the password of the selected user upon correct input validation

void viewStatistics()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	All
Description	Display some statistics for each users

void cancel()	
Input	None
Complexity	Simple
Category	Update
Trigged / called by	All
Description	Cancels any given changing operation

3.5.7 HTML Package

This packages provides functions to generate html output to the browser. It is designed with the idea of easier maintenance when working with different web-browsers. The system is designed to work with the browsers: Opera 6, Netscape 6.2 and Internet Explorer 6.0.

Class description

The package is divided into 5 classes which is shown in figure 3.10 and described in table 3.11

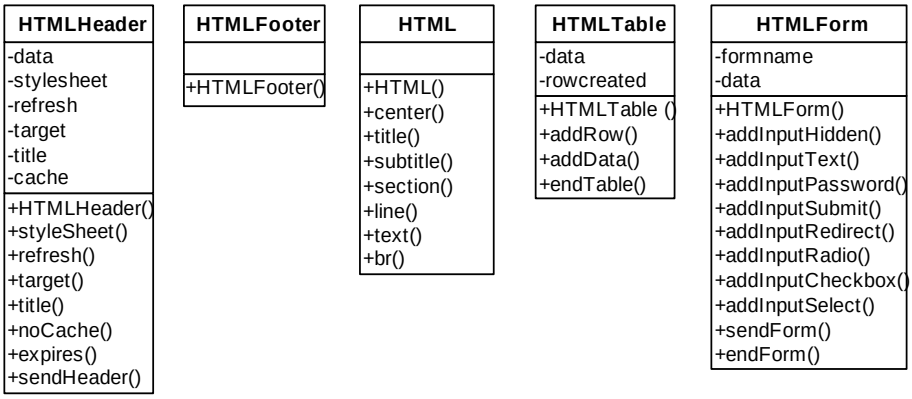


Figure 3.10 Class diagram of the HTML package

Class name	Description
HTMLHeader	This class is responsible for sending a default HTML-document header to the browser. It guaranties that only one header is sent to the browser and that it always is sent
HTMLFooter	This class send the default HTML document footer. It guaranties that only one footer is sent to the browser and that it always is sent
HTML	This class provides different functionalities to draw items in a HTML document
HTMLTable	This class provides functionality to draw tables on the webpage
HTMLForm	This class provides functionality to generate HTML-Forms

Table 3.11 Class description of the HTML package

The different functions in the classes is not described in this document.

4 Test Results

Selected test scenarios are presented in the following sections.

4.1 Use Case Test

A test scenario for use case validation consists of a number of successive function calls. Figure 4.1 shows such a scenario.

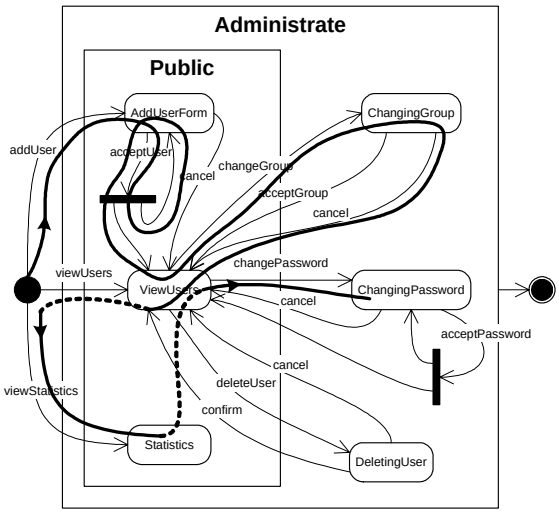


Figure 4.1 Scenario for use case test

Starting with a call of the function addUser a sequence of five legal calls is performed, ending up in the state ViewUsers. The dashed line exiting this state represents the call of the function viewStatistics, which may be executed from any state. In the state obtained after execution a call of the function changePassword is tried – depicted by the second dashed line to the ViewUser state. This should result in an illegal state transition error.

The result of the test is shown in table 4.1, which contains relevant entries singled out from the log. As expected, the first six function calls were carried out successfully while the last one was detected as illegal.

Message
State changed correct (Welcome -> AddUserForm)
State changed correct (AddUserForm -> ViewUser)
State changed correct (AddUserForm -> ViewUser)
State changed correct (ViewUser -> ChangeGroupForm)
State changed correct (ChangeGroupForm -> ViewUser)
State changed correct (ViewUser -> Statistics)
State changed incorrect (Statistics -> ChangePasswordForm) Required: ViewUser

Table 4.1 Log results from use case test

4.2 User Validation Test

To test the user validation three types of scenarios are needed. First, a member of the MCS calls a function related to the flight plan. Because MCS is, the only user group allowed accessing the flight plan this should be legal. As expected, the log shows that the user group is correct for the function call.

Name	Value
Class	FlightPlan
Function	viewFlightPlan
Message	User group is correct (Group required: MCS)
Group	MCS

Table 4.2 Log result from user validation

Next, an example of a function call that should not be allowed by the validation system is considered. The scenario in this case is a MCS trying to administrate the users of the system. Like before the log in table 4.3 shows that the unintended use of the system is avoided by the validation system.

Name	Value
Class	Administrate
Function	viewUsers
Message	User group is incorrect (Group required: Admin)
Group	MCS

Table 4.3 Log result from user validation

Finally, it is tested whether or not the limitation on the number of active group members is respected. The scenario for this test is as follows: One MCS is logged on and another tries to get access. In the case of a distributed control room for a small-satellite, this situation is avoided by informing the user about the problem and granting the user access as a lower prioritized group. table 4.4 shows that the validation system takes care of this situation.

Name	Value
Class	Logon
Function	dologon
Message	Group has changed (Mission Control Staff -> Scientific Staff)

Table 4.4 Log result from user validation

4.3 Secure Communication Test

Test of the encryption of the communication is divided into four scenarios. Each scenario is specified by the needs for and the presence of encrypted communication. The scenarios are represented in table 4.5 along with the results of the test. The test result fulfilled the expectations

Class	Function	SSL Needed	SSL Present	Access	Result
Administrate	viewUsers	Y	Y	Y	Communication is correctly (SSL required: Y)
Administrate	viewUsers	Y	N	N	Communication is incorrect (SSL required: Y)
Image	Preview	N	Y	Y	Communication is correctly (SSL required: N)
Image	Preview	N	N	Y	Communication is correctly (SSL required: N)

Table 4.5 The encryption communication scenarios and the test results

Appendix I Protocol: HTTP (Hypertext transfer protocol)

Abstract:

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic stateless protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

Type	Description
Platform independencies	Yes
Real time requirement	None
Security	No support
Proxy compatible	HTTP-Compatible proxy
Secure send / receive acknowledgement	Controlled by lower protocols
Broadcast compatible	No.
Combinable	Yes. Many known protocols is combined with this protocol
Dependencies	None
Type of protocol	Transfer protocol
Media	ACSII

References

http://www.ietf.org/rfc/rfc2616.txt

Appendix II Protocol: RMI

Abstract:

The design goal for the RMI architecture was to create a Java distributed object model that integrates naturally into the Java programming language and the local object model. RMI architects have succeeded; creating a system that extends the safety and robustness of the Java architecture to the distributed computing world.

Type	Description
Platform independent (Windows, Linux)	Yes – as platform independent as Java itself.
Real time requirement	No
Security (Does it support it)	Yes - using a security manager guarantees that classes are loaded from a “trusted” source.
Proxy compatible (HTTP)	Yes - The RMI call data is sent outside as the body of an HTTP POST request, and the return information is sent back in the body of the HTTP response.
Secure send / receive acknowledgement (TCP/UDP)	Uses TCP, but UDP could be implemented.
Broadcast compatible	Yes - With Java MulticastSocket
Combinable	
Dependencies	The current transport implementation is TCP-based (using Java sockets), but a transport based on UDP could be substituted [1) System Architecture - Overview].
Type of protocol (Data transfer specification)	Data protocol
Media (ASCII/BINARY)	

Reference

http://engronline.ee.memphis.edu/AdvJava/Lectures/rmi_spec.htm

http://developer.java.sun.com/developer/onlineTraining/rmi/RMI.html#IntroRMI

Appendix III Protocol: RPC (Remote Procedure Call)

Abstract:

This document specifies a message protocol used in implementing Sun's Remote Procedure Call (RPC) package. The message protocol is specified with the eXternal Data Representation (XDR) language [9]. This document assumes that the reader is familiar with XDR. It does not attempt to justify RPC or its uses. The paper by Birrell and Nelson [1] is recommended as an excellent background to and justification of RPC.

Type	Description
Platform independencies	Yes
Real time requirement	None
Security	Yes, Using authentication protocols white DES encrypting.
Proxy compatible	No
Secure send / receive acknowledgement	Yes, using TCP protocol
Broadcast compatible	No
Combinable	None
Dependencies	
Type of protocol	Authentication message protocol
Media	Binary

References

ftp://ftp.isi.edu/in-notes/rfc1050.txt

Appendix IV Protocol: SOAP-Protocol

Abstract:

SOAP version 1.2 is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of four parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, a convention for representing remote procedure calls and responses and a binding convention for exchanging messages using an underlying protocol. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and the experimental HTTP Extension Framework.

Type	Description
Platform independencies	Yes
Real time requirement	None
Security	XML Encryption • Elementvise encryption • Document encryption
Proxy compatible	Yes through lower protocols e.g. HTTP protocol
Secure send / receive acknowledgement	Yes through lower protocols e.g. TCP protocol
Broadcast compatible	No
Combinable	None
Dependencies	HTTP Protocol
Type of protocol	Data protocol
Media	ASCII

References

http://www.w3.org/TR/2001/WD-soap12-20010709/

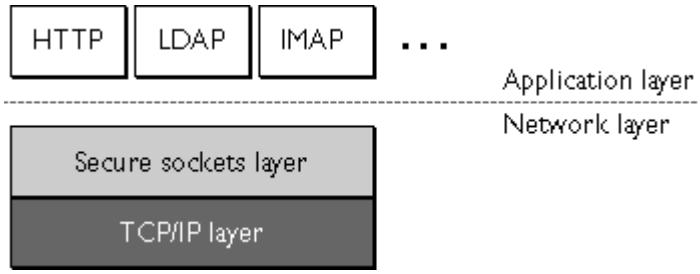
Appendix V Protocol: SSL-Protocol

Abstract:

This document specifies Version 3.0 of the Secure Sockets Layer (SSL V3.0) protocol, a security protocol that provides communications privacy over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Type	Description
Platform independencies	Yes
Real time requirement	None
Security	Provide a secure channel, controlled with server certificates and maybe client certificates
Proxy compatible	Yes. Known proxies is HTTPS
Secure send / receive acknowledgement	Controlled by lower protocols
Broadcast compatible	No. The protocol is specific designed to peer to peer connections
Combinable	Yes. A known protocol is the HTTP Protocol
Dependencies	None
Type of protocol	Security protocol
Media	BINARY

Place in OSI model



References

ftp://ftp.isi.edu/in-notes/rfc2104.txt

http://docs.iplanet.com/docs/manuals/security/sslin/

Appendix VI Protocol: VPN (Virtual Private Network)

Abstract:

VPNs offer enterprise-scale connectivity deployed on a shared infrastructure with the same policies enjoyed in a private network. These policies include security, prioritization, reliability, and end-to-end management. A VPN can be deployed over the Internet or built on a service provider's existing IP, Frame Relay, or ATM infrastructure.

VPNs based on IP can naturally extend the ubiquitous nature of intranets---over wide-area links, to remote offices, to mobile users, or to telecommuters. Further, they can extend extranets to communities of interest outside the organization linking business partners, customers, and suppliers, to provide better customer satisfaction, market differentiation, and reduced manufacturing costs.

The three basic types of VPNs are Access VPNs, Intranet VPNs, and Extranet VPNs. Access VPNs appeal to a highly mobile work force, handling remote-access connectivity for mobile users, telecommuters, and small offices through a broad range of technologies

Type	Description
Platform independencies	Yes
Real time requirement	None
Security	Yes, Using DES encrypting.
Proxy compatible	Yes, using HTTP protocol
Secure send / receive acknowledgement	Yes, if TCP protocol is used
Broadcast compatible	Yes
Combinable	All types of transport protocols e.g. HTTP, TCP, UDP
Dependencies	“HTTP”
Type of protocol	Data protocol
Media	Binary

References

http://www.cisco.com/warp/public/cc/cisco/mkt/servprod/dial/justify/profiles/avpnn_bc.htm

http://www.cisco.com/warp/public/cc/sol/mkt/ent/vpne/tech/qsvpn_wp.htm

http://comet.columbia.edu/research/architectures/VPN.html

Appendix VII Protocol: XML-Protocol

Abstract:

The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML.

Type	Description
Platform independencies	Yes
Real time requirement	None
Security	XML Encryption • Elementvise encryption • Document encryption
Proxy compatible	Yes through lower protocols e.g. HTTP protocol
Secure send / receive acknowledgement	Yes through lower protocols e.g. TCP protocol
Broadcast compatible	Depending on the lower protocols.
Combinable	No.
Dependencies	None
Type of protocol	Data protocol
Media	ASCII

Reference

http://www.w3.org/TR/2000/REC-xml-20001006

http://www.w3.org/TR/xmlenc-core/

Appendix VIII Protocol: XMLRPC (Remote Procedure Call over XML protocol)

Abstract:

It's a spec and a set of implementations that allow software running on disparate operating systems, running in different environments to make procedure calls over the Internet.
It's remote procedure calling using HTTP as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible, while allowing complex data structures to be transmitted processed and returned.

Type	Description
Platform independencies	Yes
Real time requirement	None
Security	Yes, Authentication when RPC connects are made
Proxy compatible	Yes, over HTTP protocol
Secure send / receive acknowledgement	No
Broadcast compatible	No
Combinable	None
Dependencies	XML and HTTP
Type of protocol	Message protocol
Media	Text

References

http://www.xmlrpc.com/