Horus
IMSETY
Software Requirements Document
Version 1.0  24th April 2007

Project team:  Jeroen Keiren  0569081
             Frank Koenders  0575629
             Thijs Nugteren  0574426
             Joeri de Ruiter  0578312
             Stijn Stiefelhagen  0579816
             Carst Tankink  0569954
             Pim Vullers  0575766
             Freek van Walderveen  0566348

Project manager:  Egbert Teeselink

Senior management:  L. Somers  TU/e (HG 7.83)
                    M. v.d. Brand  TU/e (HG 7.44)
Adviser:  R.J. Bril  TU/e (HG 5.09)
Customer:  E. v. Breukelen  ISIS

Computer Science, Eindhoven University of Technology, Eindhoven
Abstract

This document contains the software requirements for IMSETY. These requirements are a developer’s view of the system and contain a logical model of a system satisfying the capability requirements found in the URD [3]. Next to this model several extra-functional requirements are provided, formulating demands not easily captured in a model.
# Contents

1 Introduction  ........................................... 1  
  1.1 Purpose ............................................. 1  
  1.2 Scope .............................................. 1  
  1.3 List of definitions  ................................ 1  
  1.4 List of references  .................................. 1  
  1.5 Overview .......................................... 2  

2 General description  ................................... 3  
  2.1 Relation to current projects  ....................... 3  
  2.2 Relation to predecessor and successor projects  .... 3  
  2.3 Function and purpose  ................................ 3  
  2.4 Environment ........................................ 3  
  2.5 Relation to other systems  .......................... 4  
  2.6 General constraints  ................................ 4  
    2.6.1 Satellites .................................... 4  
    2.6.2 End users ..................................... 4  
  2.7 Model description  .................................. 4  
    2.7.1 Class model ................................... 4  
    2.7.2 Controller ..................................... 6  
    2.7.3 Login flow .................................... 8  
    2.7.4 Payload life cycle ............................. 8  
    2.7.5 Experiment workflow ............................ 9  
    2.7.6 Interactions ................................... 13  
    2.7.7 User class access .............................. 17  
  2.8 Technical risks ..................................... 19  

3 Specific requirements  ................................ 20  
  3.1 Functional requirements  ............................. 20  
    3.1.1 Account ....................................... 20  
    3.1.2 Satellite ....................................... 21  
    3.1.3 Payload ........................................ 22  
    3.1.4 Command ....................................... 23  
    3.1.5 Experiment ..................................... 23  
    3.1.6 Queue .......................................... 25  
    3.1.7 Booking ........................................ 25  
    3.1.8 Data ........................................... 25  
    3.1.9 Thumbnail ....................................... 26  
    3.1.10 High-Volume ................................... 26  
    3.1.11 Measurement ................................... 27  
    3.1.12 Plugin ......................................... 27  
    3.1.13 Preprocessor ................................... 27  
    3.1.14 Postprocessor ................................. 27
## CONTENTS

3.1.15 Representer ................................................. 27  
3.1.16 Picture .................................................... 28  
3.1.17 Video ....................................................... 28  
3.1.18 Representation ............................................ 28  
3.1.19 Observation ............................................... 28  
3.1.20 CommunicationLog ....................................... 29  
3.1.21 MCS ......................................................... 29  
3.1.22 MCC ......................................................... 30  
3.1.23 Controller .................................................. 31  
3.2 Extra-functional requirements ............................... 32  

4 Requirements traceability matrices .......................... 34  
4.1 UR to SR ......................................................... 34  
4.2 SR to UR ......................................................... 36  

A Prototype description ......................................... 39  
A.1 Rights .......................................................... 39  
A.2 Main view ....................................................... 40  
A.3 File .............................................................. 41  
A.3.1 Login ........................................................ 41  
A.3.2 Preferences ............................................... 41  
A.4 Experiment ..................................................... 41  
A.4.1 Create an experiment .................................... 42  
A.4.2 Edit experiment ........................................... 42  
A.4.3 Schedule experiment ..................................... 43  
A.5 Observe experiment .......................................... 44  
A.6 Observation data ............................................ 45  
A.6.1 Manage observers ....................................... 46  
A.7 Administration ............................................... 46  
A.7.1 Manage accounts ........................................ 46  
A.7.2 Manage satellites ......................................... 47  
A.7.3 Manage payloads ......................................... 47  
A.7.4 Communication logs ..................................... 47  

B External interfaces ............................................... 49  
B.1 The GENSO Mission Control Client ....................... 49  
B.1.1 Booking .................................................... 50  
B.2 The satellite’s Mission Control Software ................. 50  
B.2.1 Experiment / Payload specific commands ............. 50  
B.2.2 Data handling (communication & management) ....... 50
### Document status sheet

<table>
<thead>
<tr>
<th>Document title</th>
<th>Software Requirements Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document identifier</td>
<td>IMSETY/doc/SRD</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Jeroen Keiren, Frank Koenders, Thijs Nugteren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Document status</td>
<td>Externally approved</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author(s)</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>05-03-2007</td>
<td>Joeri de Ruiter</td>
<td>Created document</td>
</tr>
<tr>
<td>0.1 (Revision 473)</td>
<td>25-03-2007</td>
<td>Jeroen Keiren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
<td>First version for internal review</td>
</tr>
<tr>
<td>0.2 (Revision 505)</td>
<td>26-03-2007</td>
<td>Jeroen Keiren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
<td>First version for external review</td>
</tr>
<tr>
<td>0.3 (Revision 630)</td>
<td>02-04-2007</td>
<td>Jeroen Keiren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
<td>Version for second internal review</td>
</tr>
<tr>
<td>0.4 (Revision 675)</td>
<td>03-04-2007</td>
<td>Jeroen Keiren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
<td>Version for second external review</td>
</tr>
<tr>
<td>0.5 (Revision 746)</td>
<td>10-04-2007</td>
<td>Jeroen Keiren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
<td>Version for external acceptance by customer.</td>
</tr>
<tr>
<td>0.6 (Revision 790)</td>
<td>13-04-2007</td>
<td>Jeroen Keiren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
<td>Last changes for definitive external acceptance.</td>
</tr>
<tr>
<td>1.0 (Revision 790)</td>
<td>24-04-2007</td>
<td>Jeroen Keiren, Joeri de Ruiter, Stijn Stiefelhagen, Carst Tankink, Pim Vullers</td>
<td>No changes.</td>
</tr>
</tbody>
</table>
Document change report

<table>
<thead>
<tr>
<th>Document title</th>
<th>Software Requirements Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document identifier</td>
<td>IMSETY/doc/SRD</td>
</tr>
<tr>
<td>Date of changes</td>
<td>10-04-2007</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section number</th>
<th>Reason for change</th>
</tr>
</thead>
</table>
Chapter 1

Introduction

1.1 Purpose

The software requirements document (SRD) provides a translation of all specific requirements found in chapter three of the URD \[3\] into software requirements. In contrast to the user requirements, explicitly describing the customer’s wishes, these software requirements represent the developers’ view of what the system must be able to do.

Specifically, the software requirements dictate—in developer’s terms—what IMSETY must do, and not how (in other words, it is implementation-independent). These requirements are modeled in a logical model, which provides a simplified view of the system’s content and behaviour.

The SRD represents a negotiated contract between Horus and ISIS. All of the listed requirements, and only these, will be implemented in IMSETY, according to their priorities. Any changes to these requirements require the full consent of both parties.

1.2 Scope

IMSETY will provide scientists and observers with an easy to use program interface to conduct experiments on space based samples and possible reference samples on Earth.

1.3 List of definitions

This section contains the definitions of all used terms, acronyms and abbreviations in this document.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESA</td>
<td>European Space Agency</td>
</tr>
<tr>
<td>GENSO</td>
<td>Global Educational Network for Satellite Operations</td>
</tr>
<tr>
<td>MCC</td>
<td>Mission Control Client, see the URD [3] for more information.</td>
</tr>
<tr>
<td>MCS</td>
<td>Mission Control Software, see the URD [3] for more information.</td>
</tr>
<tr>
<td>MSC</td>
<td>Message Sequence Chart, a chart used to model communications with the system by actors and communication within the system by separate objects.</td>
</tr>
<tr>
<td>SRD</td>
<td>Software Requirements Document</td>
</tr>
<tr>
<td>URD</td>
<td>User Requirements Document</td>
</tr>
</tbody>
</table>

1.4 List of references


1.5 Overview

The remainder of this document is divided into three chapters, as dictated by the ESA Software Engineering Standards [1]: General description (chapter 2), Specific requirements (chapter 3) and Traceability matrices (chapter 4).

The general description gives a simplified overview of the system. This description is broken down in the following parts:

- the relation of IMSETY to current projects that might influence it,
- the relation of IMSETY to previous and possible future projects,
- the function and purpose of the system,
- a list of the systems with which IMSETY will interact and their essential characteristics,
- the general constraints for IMSETY, and
- a logical model of IMSETY, with a description clarifying its components.

The logical model consists of a class model, describing the entities that should exist in the system —if it is to fulfil the user requirements—, and the actions that can be executed upon them.

To describe requirements that can be more easily captured with state diagrams, several workflows, modeled as Petri nets, are provided. These models are also described.

The specific requirements are subdivided into functional requirements, which are distilled from the specific requirements from the URD [3], and extra-functional requirements, describing those requirements that cannot easily be captured in a model.

The traceability matrices relate the specific requirements found in the URD to those in the SRD and vice versa.

Two appendices appear at the end of this document. The first describes the prototype created for IMSETY’s GUI. This prototype should be considered an integral part of the software requirements, illustrating those requirements that are not easily caught in a model. The second appendix describes the interfaces IMSETY has to the two other types of systems in its deployment environment: MCSes and an MCCs.
Chapter 2

General description

2.1 Relation to current projects

This project is closely related to the Global Educational Network for Satellite Operations (GENSO) project. GENSO is currently under development and will not be finished before IMSETY’s delivery deadline. GENSO’s goal is to abstract from separate ground stations and, by connecting individual ground stations, increase the number of communication opportunities with satellites launched for educational purposes.

2.2 Relation to predecessor and successor projects

There are no predecessors to the IMSETY project. This is because ISIS explores a new avenue of opportunities for scientists, to be able to book small missions, and there has been no need for interfaces like IMSETY.

Since IMSETY will provide a plug-in system, future projects could use IMSETY’s framework to develop plug-ins for their specific data or experiment types. At this time there are no known parties, apart from ISIS, which are interested to make use of IMSETY’s functionality.

2.3 Function and purpose

IMSETY will provide an easy to use interface for scientists, observers and administrators. It will abstract from technical details so users with no technical knowledge will still be able to use it. The system should allow the plugging in of (sub)systems for pre-, and postprocessing and the representation of specific data types.

2.4 Environment

The interface of the system shall be accessible on Windows 2000/XP/Vista, Mac OS X, Sun Solaris and Linux. IMSETY’s backend shall run on a Linux server as described in the URD [3].

The users of the system are:

- Administrators
- Observers
- Scientists

Their characteristics can be found in section 2.4 of the URD [3].
2.5 Relation to other systems

IMSETY shall communicate with a satellite using an MCC and an MCS, which in turn also uses
the MCC to abstract from several ground stations for more communication opportunities. This
communication consists of uploading experiments and commands and downloading requested data.
The MCC is provided by GENSO and is used by IMSETY for the booking of ground stations.

The MCS is a system tailor-made by ISIS for one specific satellite. This MCS shall receive
commands from IMSETY and compile them in a format understandable by the satellite. It then
uses the MCC to send the compiled command to the satellite and waits for an acknowledgement.
If the satellite confirms that it has received the correct command, the MCS shall inform IMSETY
of this and enable the command on the satellite. If this is not the case, the MCS shall inform
IMSETY of this fact and take no action. Additionally, the MCS is used by IMSETY to download
data from the satellite.

2.6 General constraints

2.6.1 Satellites

The IMSETY project will communicate with satellites and since a satellite is in orbit over the
Earth, there is no constant communication with the satellite. This leads to communicating with
the satellite only on those moments when it passes a ground station.

Furthermore, communication with a satellite is potentially unreliable. Therefore, the system
should be able to cope with unexpected interruptions in satellite communication.

2.6.2 End users

The end users do not need to have no technical knowledge on satellite communication, therefore
the IMSETY project will abstract from technical details as much as possible and offer the users
an easy to use interface.

2.7 Model description

2.7.1 Class model

This section describes the class model depicted in figure 2.2. In its description, classes from the
domain model found in figure 2.3 of the URD [3] are emphasized. In the model, any connections
between the classes not used in formal definitions do not have a label, to keep the model as clean
as possible. For the same reason, we do not show attributes which are not directly needed in the
formulation of requirements satisfying the URD [3].

User management

To use the system, a User has to have a registered account with the system. This account is
represented by the Account class in the model. This class has connections to both the Payload class
and the Experiment class, since Observers should be able to observe experiments and Scientists
should be able to access their payloads, including the possibility to create and observe experiments
on these payloads. Furthermore, scientists can observe experiments of other scientists, if granted
access to these.

In order to express that a scientist should at least be able to observe the experiments on his
own payloads, we introduce the following constraint:

\[
(\forall a \in \text{Account} :: (\forall p \in \text{Payload} : p \in a.\text{hasAccessTo} : p.\text{definedOn} \subseteq a.) ) \quad (2.1)
\]
CHAPTER 2. GENERAL DESCRIPTION

The Payload, the Command and the Experiment

The Payload class represents the Payload from the domain. This payload is linked to the satellite in which it is physically installed.

In order to represent Experiments, an Experiment class is provided. An experiment consists of a list of Commands, which are gathered from the Command class coupled to a payload. To express that the commands of these experiments need to be predefined per payload, we introduce the following constraint:

\[(\forall e \in \text{Experiment} :: (\forall c \in e.\text{listOfCommands} :: c \in e.\text{definedOn}.\text{isAvailable}))\]  

(2.2)

Satellites and communications

Satellite is a class representing Satellites which have zero or more payloads linked to them. Only one experiment can be executed at a time on one satellite. In predicate logic:

\[(\forall s \in \text{Satellite} :: (\forall e_1, e_2 \in \text{Experiment} : e_1 \neq e_2 \land e_1.\text{definedOn}.\text{Contains} = s \land e_2.\text{definedOn}.\text{Contains} = s : \{t \in \text{time} | e_1.\text{scheduledFrom} \leq t \leq e_1.\text{scheduledTo}\} \cap \{t \in \text{time} | e_2.\text{scheduledFrom} \leq t \leq e_2.\text{scheduledTo}\} = \emptyset\})\]  

(2.3)

To communicate with the physical satellite, IMSETY communicates with the MCC and the MCS through the communication classes in the model. These communications are also logged. Downloading and uploading of data is done by adding requests for these items to the queue. This leads to bookings, as described in the message sequence charts in section 2.7.6.

Observations and data

Observations are represented by the Observation class of the model. These observations lead to Data, which can be one of several types. Data can only be gathered during an experiment:

\[(\forall d \in \text{Data} :: (\exists e \in \text{Experiment} : d.\text{From}.\text{Produces} = e : e.\text{scheduledFrom} \leq d.\text{gatherAt} \leq e.\text{scheduledTo}))\]  

(2.4)

Data is a generalization of the data types Thumbnail, High-Volume data and Measurements. The Thumbnail class represents the small thumbnails downloaded from the satellite. By marking these thumbnails, it is possible to make a selection, request the download or request the deletion of High-Volume data. A measurement is different from High-Volume data in that it can always be received, without the need for a thumbnail.

Since Data is a generalization of these three classes, the methods and attributes defined on the Data class are also defined on High-Volume, Thumbnail and Measurement.

Plug-ins: Preprocessor, Representer, Postprocessor

To sort and represent data, IMSETY provides plug-in systems. A common facility for all these plug-ins is the possibility to configure them. A Preprocessor transforms raw data into a format which can be used by IMSETY’s Representers. These representers create a Representation, which can be shown through the user interface. Those representers planned to be released with IMSETY are a video and a picture representer, which are therefore shown in the model. To modify what he sees on screen, the user can apply a Postprocessor to the representation. This leads to a new representation, which can then be shown by the interface.
2.7.2 Controller

In order to shield the user from several inter-object communications as well as some communications to the outside world, we provide the controller facade to the system depicted in figure 2.1. The actions taken by the controller are shown in section 2.7.6. From the diagrams in that section, one can see that the controllers calls several functions of the other classes in the model.

![Controller diagram]

Figure 2.1: A controller facade to the system
CHAPTER 2. GENERAL DESCRIPTION

Figure 2.2: A class model of the entities
2.7.3 Login flow

We describe some parts of the process in the Petri net formalism. We will give a short, informal description of this formalism here. A Petri net consists of places (the circular components) and transitions (the square components). In the places a token (depicted by a dot) can be present, which depicts that the system is in a current state. A transition can account for a change of state of the system whenever all places of which the transition has an incoming arc have got a token. The transition will then place a token in all places which it has got an outgoing arc to. Incoming and outgoing arcs are presented by an arrow (an arrow can have arrowheads on both ends, which denotes both an incoming and an outgoing arc). Furthermore there are inhibitor arcs, these are depicted by a line with a dot at the transition end, this means that the transition can not fire as long as there are tokens in the place it is connected to through the inhibitor arc. A more formal description of the formalism can be found in [5].

![Figure 2.3: The login flow](image)

In figure 2.3 the login flow of the system is shown. Users first have to log in before they can perform any action in the system. When the user is done, he can logout. An account can be used by at most 1 user at the same time.

2.7.4 Payload life cycle

![Figure 2.4: A payload life cycle](image)

In figure 2.4 the life cycle of a payload is shown. At any time only one user can modify the data describing a payload or compose an experiment on the payload. Only an administrator can create or modify a payload. Both a scientist and an administrator can compose an experiment on a payload.
CHAPTER 2. GENERAL DESCRIPTION

2.7.5 Experiment workflow

The key part of the system is the experiment. In the URD [3], an abstract workflow of the experiment has been provided. We will now elaborate on this further, and create a more complex model, which resembles reality. For convenience we repeat the model given in the URD here in figure 2.5.

Figure 2.5: Workflow which depicts the phases an experiment goes through in a very abstract way

In fact there are two different components involved in the experiment process. One which is part of IMSETY and one which runs on the satellite. Both of these resemble some part of the simple workflow as described in the URD, and some information overlaps between the two components. Note that this overlap is because the state of the experiment should be known locally, which would be impossible without overlap because of the unreliable and restricted communication between the system and the satellite. In figure 2.6 we describe the component of the experiment model in IMSETY is given. In figure 2.8 the component of the experiment model which is situated on the satellite can be found. Note that the creation and editing parts of the simple model from figure 2.5 are situated in the IMSETY part of the experiment, whereas the rest of the simple model can be found in both components for previously described reasons. The creation of an experiment is further described in figure 2.7 (this subnet is the same for all model that will be described further in this section, except for this final version of the experiment model).

Figure 2.6: The IMSETY component of the experiment

The two components as described until now are useless unless we define some communication between them. If we add the minimal amount of communication to these two components we obtain the model as in figure 2.9. Note that in this model the edit functionality is not yet fully modeled for simplicity reasons, and that an extra transition “remove superfluous commands” has been added in order to describe the execution of all sent commands, this is needed because of the asynchronous nature of communication. It is also good to observe that the process can only finish once all extra commands have been executed.

Taking a closer look to the model in 2.9 it says that initially an experiment will have to be
create experiment compose finish composing schedule queued for upload
window of opportunity for upload later than scheduled start
composing composed queued for uploadscheduled
experiment created

Figure 2.7: Subnet for the creation of an experiment in figure 2.6

Figure 2.8: The satellite component of the experiment

created by the scientist. When created it can be uploaded to the satellite. Once the experiment has been transmitted to the satellite its execution will be started automatically at the scheduled time. In the mean time, the experiment can still be edited in IMSETY. Also, once execution of the experiment has been started on the satellite, and there is a window of opportunity for communication available, the scientist can intervene with the experiment by sending commands. Before these commands can be sent, the experiment needs to be stopped, to prevent commands running out of the window. After the experiment has finished the pending commands are executed, and the system will terminate cleanly.

As said before, so far we have not gone into detail with respect to the editing of an already uploaded experiment. We could first extend the “edit” transition in the IMSETY component, only extending this locally however is not enough in order to make the satellite execute the newly defined experiment. In order to upload the new experiment we have to add some extra communication. The model in which the editing capabilities as well as the communication needed for it is depicted in figure 2.10. The difference between this model and the model in figure 2.9 is that in this model the edited experiment can be reuploaded to the satellite.

However, there are some problems with the newly created model in which all communication and editing functionality has been incorporated. Some questions one could ask are not yet fully satisfied in this model. For example, what happens to the old experiment on uploading a new one? If it has not yet been started it should be overwritten, if it was already started the upload of the new experiment should fail, and the original experiment should be preserved. To also cope with these kinds of problem, some additions to the model have been done. The model in figure 2.11 guarantees that once the process finishes, there always is just one experiment known to the system, this is the so called soundness property of the system. Note that even in this model there is a risk with respect to communication. Between the satellite and IMSETY here should be GENSO and the MCS via which all communication with the satellite is done. These systems should take care of any communication unreliability problems. However it will always be possible
that an upload fails, but when this occurs we expect to be notified about this failure by the MCS. Because of these changes there has also been a modification to the create experiment subnet, this is represented in its new form in figure 2.12.
Figure 2.11: An experiment workflow
CHAPTER 2. GENERAL DESCRIPTION

2.7.6 Interactions

This section describes the high level interactions with and within IMSETY through message sequence charts (MSCs). A description of the semantics of MSCs can be found in a paper published as TU/e Computer Science report CSR00-12 [4].

The interaction with the system is depicted by messages sent by an actor to one of the objects in the system. Interactions within the system are described by message going back and forth between objects of the system.

Queue processing

The Queue processing MSC depicted in figure 2.13 shows the messages sent after a request for upload or download has been added to the queue.

After this entry has been added, the system invokes the MCC object to initiate a booking of a pass. The MCC object in IMSETY then requests a list of opportunities from the actual MCC, and creates a booking for an appropriate moment. Since not all passes are available for uploading the data, the appropriate moment in this case is the earliest moment at which upload can occur. Confirmation of the booking is communicated to the queue. If a booking is rejected, the system will try to book another moment.

After receiving a booking acknowledgement, the system retrieves the booking information. At this booked moment, depending on the type of communication, the queue instructs the MCS object to initiate either the download or the upload. The functions used in this MSC are abstractions of the actual functions used to communicate with the MCS.

The path for downloading is more complicated than uploading, since the raw data collected from the satellite needs to be processed and stored by a preprocessor. Note that when the raw data is stored, a message is sent to the user, not when it is preprocessed and afterwards stored.
CHAPTER 2. GENERAL DESCRIPTION

Experiment construction

The MSC in figure 2.14 describes the interactions started when a user creates an experiment. The construction of an experiment is a request made by the user to the controller facade, which in turn creates an experiment and requests a list of commands, that belong to the payload for which the user creates an experiment.

Choosing from these commands, the user can iteratively add, modify or delete commands, until he is satisfied. Then, he can schedule or reschedule the experiment. For this schedule, the system tries to create a booking. The result of this attempt is communicated to the user. When the scheduling is confirmed, the experiment is added to the queue to be uploaded, following which the queue processing interaction is started.
Downloading

The MSC in figure [2.15] shows messages sent when the user requests the download of high volume data.

First, the user requests a list of thumbnails, which are shown by the representer. The user then marks the thumbnails he finds interesting for download. The marked items are then added to the
queue. Note that this model assumes that only those thumbnails can be marked for which high-volume data is available on the satellite. If this was not the case, the download cannot continue, and the MCS should return an error to the system.

**Figure 2.15: Downloading**

**Experiment observation**

The MSC in figure 2.16 describes the interaction initiated when a user starts to observe an experiment. It is assumed this MSC is executed during a two-way booking, meaning that an MCS has all the access it needs to receive data from the satellite.

First of all, note that this MSC is similar to the downloading MSC. The main difference is the absence of the queue in the observation chart, and the showing and postprocessing of data representations.

The postprocessing assumes that a list of postprocessors has been defined by the user, which are executed in turn on the representation. This is done in a “pipes and filters” manner, meaning the second postprocessor processes the output of the first postprocessor, the third processes that of the second postprocessor and so on.

**Figure 2.16: Observing**
CHAPTER 2. GENERAL DESCRIPTION

Experiment intervention

The MSC in figure 2.17 describes the actions taken when a user wants to intervene in the execution of an experiment. The controller gathers the list of possible commands. From these commands, the user picks one command, which is sent to the satellite by the MCS. The status of this command is stored in the system.

![Figure 2.17: Intervention](image)

2.7.7 User class access

Table 2.1 describes the access rights each user class has. This table mentions all methods (grouped per object class) of the class model, and shows a tick (✓) if a user class is allowed to use this method, or a cross (✗) if it is not. In the case of the scientist and observer classes, a tick means that a user belonging to one of these user classes is only allowed to access this function on instances of the object class to which he is linked. A method which no users are allowed to access is a method which only the system is able to invoke.

<table>
<thead>
<tr>
<th></th>
<th>Administrator</th>
<th>Scientist</th>
<th>Observer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>create</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>register</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>retrieve</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>update</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>unregister</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>addPayload</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>removePayload</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>addExperiment</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>removeExperiment</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

Satellite

<table>
<thead>
<tr>
<th></th>
<th>Administrator</th>
<th>Scientist</th>
<th>Observer</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>retrieve</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>update</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>unregister</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Payload

<table>
<thead>
<tr>
<th></th>
<th>Administrator</th>
<th>Scientist</th>
<th>Observer</th>
</tr>
</thead>
<tbody>
<tr>
<td>register</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>retrieve</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>update</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
### CHAPTER 2. GENERAL DESCRIPTION

<table>
<thead>
<tr>
<th>Command</th>
<th>unregister</th>
<th>link</th>
<th>getListOfCommands</th>
</tr>
</thead>
<tbody>
<tr>
<td>define</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>link</td>
<td>√</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>retrieve</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Experiment</th>
<th>retrieve</th>
<th>schedule</th>
<th>unschedule</th>
<th>add</th>
<th>modify</th>
<th>remove</th>
<th>upload</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieve</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>schedule</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unschedule</td>
<td>√</td>
<td>√</td>
<td>×</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>add</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modify</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>upload</td>
<td>√</td>
<td>√</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thumbnail</th>
<th>markInteresting</th>
<th>markForDownload</th>
<th>markForDeletion</th>
</tr>
</thead>
<tbody>
<tr>
<td>markInteresting</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>markForDownload</td>
<td>√</td>
<td>√</td>
<td>×</td>
</tr>
<tr>
<td>markForDeletion</td>
<td></td>
<td>√</td>
<td>×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data</th>
<th>retrieve</th>
<th>export</th>
<th>store</th>
</tr>
</thead>
<tbody>
<tr>
<td>retrieve</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>export</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>store</td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Preprocessor</th>
<th>process</th>
<th>configure</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>configure</td>
<td></td>
<td>×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Postprocessor</th>
<th>process</th>
<th>configure</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>configure</td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Representer</th>
<th>show</th>
<th>showThumbnails</th>
<th>markForThumbnails</th>
<th>createRepresentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>show</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>showThumbnails</td>
<td></td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>markForThumbnails</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>createRepresentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Observation</th>
<th>getListOfThumbnails</th>
</tr>
</thead>
<tbody>
<tr>
<td>getListOfThumbnails</td>
<td>×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Queue</th>
<th>add</th>
<th>remove</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>remove</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Booking</th>
<th>create</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>×</td>
</tr>
</tbody>
</table>
2.8 Technical risks

During this development process and IMSETY’s life cycle, several threats stand in the way of the system’s success. These threats are mainly caused by the environment in which the system is to operate.

The greatest risk is caused by the development status of the GENSO project. Because the GENSO’s MCC requirements are still being drawn up, they are not stable yet, leaving the possibility open that assumptions made on these requirements may no longer be valid. To counter, IMSETY will be built as loosely coupled as possible concerning the components which interact with the MCC. This enables easy adaption to modifications in the interface. However, if functionality is removed from these interfaces (so they become more restricted to the outside world) this may have high impact on the success of IMSETY, beyond the control of Horus.

Possible risks caused by the MCSes are mainly covered by the fact that those MCSes should conform to the interface described in the Interface and Protocol Definition [2]. However it is possible that an MCS will not implement the functionality we need. Such MCSes are not compatible with IMSETY.
Chapter 3

Specific requirements

3.1 Functional requirements

In this section, the specific requirements derived from the class model in figure 2.2 are described. These requirements are grouped by class, and subdivided in a general requirement describing what the class is used for, if it is a class needed in itself to model a user requirement, an attribute list describing the attributes of each class and a methods list describing its methods.

3.1.1 Account

SRFUR1

For each user of the system the account class describes their account.

Attributes

type : enum(administrator, scientist, observer)

SRFUR2

Every account is either an administrator, observer or scientist account.

Methods

This create method is only mentioned here and not in other classes (except booking), the create method works the same for other classes.

create(p: Parameters, t: Type)

SRFUR3

Creates an account with p and t in the system, which in not yet active (i.e. registered)

Pre: true

Post: The account with p and t has been created in the system

register()

SRFUR4

Registers an account (meaning the account will be available for login) with the system.

Pre: true

Post: The account is registered with the system

retrieved()
CHAPTER 3. SPECIFIC REQUIREMENTS

SRFUR5
Retrieves the information stored in the account.
Pre: true
Return: Account information

update(p: Parameters, t: Type)

SRFUR6
Updates the account information with p and t.
Pre: true
Post: The account is updated with p and t

unregister()

SRFUR7
Unregisters the account from the system. The account is not removed from the system, but users can no longer log in on this account.
Pre: The account is registered with the system
Post: The account is unregistered from the system

addPayload(p: Payload)

SRFUR8
Gives this account the rights to access p.
Pre: The account is registered with the system
Post: The account has rights to access the payload p and all related experiments

removePayload(p: Payload)

SRFUR9
Removes the rights to access p from this account.
Pre: The account is registered with the system
Post: The account does not have rights to access the payload p

addExperiment(e: experiment)

SRFUR10
Add e to the account
Pre: true
Post: Experiment e has been added to the account

removeExperiment(e: experiment)

SRFUR11
Remove e from the account
Pre: true
Post: Experiment e is removed from the account

3.1.2 Satellite

Attributes

multipleExperiments : boolean

SRFUR12
Whether or not multiple experiments may be executed simultaneously on this satellite as described in equation 2.3
CHAPTER 3. SPECIFIC REQUIREMENTS

Methods

register()

SRFUR13
Register a satellite with the system, so it can be used within the system.
Pre: true
Post: The satellite is registered with the system

retrieve()

SRFUR14
Retrieves the satellite information.
Pre: true
Return: Satellite information

update(p: Parameters)

SRFUR15
Updates the satellite information with p.
Pre: The satellite is registered with the system
Post: The satellite is updated with p

unregister()

SRFUR16
Unregisters the satellite information.
Pre: The satellite is registered with the system
Post: The satellite is unregistered from the system

3.1.3 Payload

Methods

register()

SRFUR17
Registers payload with the system; the payload can now be used within the system.
Pre: true
Post: The payload is registered with the system

retrieve()

SRFUR18
Retrieves the payload information.
Pre: true
Return: Payload information

update(p: parameters)

SRFUR19
Updates the payload information with p.
Pre: The payload is registered with the system
Post: The payload is updated with p

unregister()
CHAPTER 3. SPECIFIC REQUIREMENTS

SRFUR20
Unregisters the payload information.
Pre: The payload is registered with the system
Post: The payload is unregistered from the system

\textbf{link}(s: \text{Satellite})

SRFUR21
Links the payload to \(s\).
Pre: The payload is registered with the system
Post: The payload is linked to satellite \(s\)

getListOfCommands()

SRFUR22
Retrieve all commands applicable to the payload.
Pre: \textbf{true}
Return: All commands applicable to the payload

3.1.4 Command

Methods

\textbf{define}(\text{command, parameters})

SRFUR23
Defines a command.
Pre: \textbf{true}
Post: The command is created and available for linking to a payload.

\textbf{link}(p: \text{Payload})

SRFUR24
Links command to \(p\).
Pre: \textbf{true}
Post: The command is linked to payload \(p\)

\textbf{retrieve}()

SRFUR25
Retrieves command information.
Pre: \textbf{true}
Return: Command information

3.1.5 Experiment

Attributes

\textbf{scheduledFrom}: \text{time}

SRFUR26
Contains the time when the experiment will start.

\textbf{scheduledTo}: \text{time}

SRFUR27
Contains the time when the experiment will finish.
CHAPTER 3. SPECIFIC REQUIREMENTS

listOfCommands: [Command]

SRFUR28
Contains commands that constitute the experiment.

Methods

retrieve()

SRFUR29
Retrieves the experiment.

Pre: true
Return: Experiment

add(c: Command, p: Parameters)

SRFUR30
Add command c, with parameters p, to the experiment.

Pre: true
Post: Command c with parameters p is added to the experiment

schedule(t: time)

SRFUR31
Schedules the experiment to be executed on t.

Pre: true
Post: The experiment is scheduled on t

unschedule()

SRFUR32
Unschedules the experiment.

Pre: true
Post: The experiment is unscheduled

remove(c: Command)

SRFUR33
Removes command c from the experiment.

Pre: Command c is included in the experiment
Post: Command c is removed from the experiment

modify(c: Command, p: Parameters)

SRFUR34
Sets command c’s parameters to parameters p.

Pre: Command c is included in the experiment
Post: Command c is updated with parameters p in the experiment

upload()

SRFUR35
Uploads the experiment to the satellite when possible.

Pre: The experiment is scheduled
Post: The experiment is added to the queue for upload to the satellite
CHAPTER 3. SPECIFIC REQUIREMENTS

3.1.6 Queue

SRFUR36
Contains the data that needs to be uploaded to or downloaded from the satellite.

Methods

**add(t: Type, d: Data)**

SRFUR37
Add d with type t to the queue.
Pre: true
Post: The data d with type t is added to the queue

**remove(d: Data)**

SRFUR38
Remove d from the queue.
Pre: d is included in the queue
Post: d is removed from the queue

3.1.7 Booking

SRFUR39
Contains a booking with GENSO’s MCC.

Attributes

type : enum(one-way, two-way)

SRFUR40
Determines whether a booking is one way or two way.

Methods

**create(ty: Type, ti: time)**

SRFUR41
By making an instance of this class, there is a booking at time ti of type ty
Pre: true
Post: Booking instance has been created

3.1.8 Data

SRFUR42
The data class contains observation data.

Attributes

gatheredAt : time

SRFUR43
Contains the data at which the data was gathered.
Methods

retrieve()

**SRFUR44**
Retrieves the observation data from the system.
*Pre:* true
*Return:* Data

export(t: Type)

**SRFUR45**
Exports the observation data to the given format t.
*Pre:* The data can be exported to format t
*Return:* The data in format t

store(d: Data)

**SRFUR46**
Stores d.
*Pre:* true
*Post:* The data d is stored

3.1.9 Thumbnail

**SRFUR47**
Contains the thumbnails provided by the satellite during an experiment.

Methods

markInteresting()

**SRFUR48**
Mark a thumbnail to distinguish it from the less interesting thumbnails.
*Pre:* true
*Post:* The thumbnail is marked interesting

markForDownload()

**SRFUR49**
Mark a thumbnail such that its high-volume data will be downloaded.
*Pre:* true
*Post:* The thumbnail is marked for download

markForDeletion()

**SRFUR50**
Mark a thumbnail for deletion so that its associated high-volume data on the satellite will be deleted.
*Pre:* true
*Post:* The high-volume data on the satellite is deleted

3.1.10 High-Volume

**SRFUR51**
The high-volume class contains high-volume versions of thumbnails.
3.1.11 Measurement

**SRFUR52**

The measurement class contains the single value (simple) observation data. Must have

3.1.12 Plugin

**SRFUR53**

Provides a plug-in system. Should have

**Methods**

`configure(p: Parameters)`

**SRFUR54**

Configures the plug-in. Should have

**Pre:** true

**Post:** The plugin is configured with parameters \( p \)

3.1.13 Preprocessor

**SRFUR55**

Provides preprocessing of experiment data entering the system. Should have

3.1.14 Postprocessor

**SRFUR56**

Provides postprocessing of data representations. Should have

3.1.15 Representer

**SRFUR57**

Provides representations of data. Could have

**Methods**

`show()`

**SRFUR58**

Show the representation. Could have

**Pre:** true

**Post:** The representation is shown

`showThumbnails()`

**SRFUR59**

Show the thumbnails. Could have

**Pre:** true

**Post:** The thumbnails are shown

`markForDownload()`
CHAPTER 3. SPECIFIC REQUIREMENTS

SRFUR60 Could have
Mark the data $d$ to which the representer is associated by the isRepresentedBy-relation (see the class model in figure 2.2) for download.

Pre: true

Post: The data $d$ is marked for download

createrepresentation()  

SRFUR61 Could have
Create a representation of the data $d$ to which the representer is associated by the isRepresentedBy-relation. (see the class model in figure 2.2)

Pre: true

Post: The representation of data $d$ is created

3.1.16 Picture

SRFUR62 Must have
Provides visualization of picture data.

3.1.17 Video

SRFUR63 Must have
Provides visualization of video data.

3.1.18 Representation

SRFUR64 Must have
The representation class contains representations created from the observation data.

Methods

store(d: Data)

SRFUR65 Could have
Store the data $d$.

Pre: true

Post: Store the data $d$ from the representation

retrive()  

SRFUR66 Could have
Retrieve the representation.

Pre: true

Return: The representation

3.1.19 Observation

SRFUR67 Must have
The observation class groups data items from an observation together.

getlistofthumbnails()
### CHAPTER 3. SPECIFIC REQUIREMENTS

SRFUR68

Must have

Provides a list of thumbnails for the current observation.
Pre: true
Return: The list of available thumbnails

---

3.1.20 CommunicationLog

SRFUR69

Must have

The communication class contains all logged communications.

#### Methods

**retrievelog()**

SRFUR70

Must have

Retrieves the logged communications.
Pre: true
Return: Logged communication

**addentry(msg: message)**

SRFUR71

Must have

Add a message to the log.
Pre: true
Post: The message has been added to the log.

---

3.1.21 MCS

SRFUR72

Must have

The MCS class provides communication with the MCSes.

#### Methods

**exec(payload: payloadIdentifier, command: string): status**

SRFUR73

Must have

Execute the given command on the payload
Pre: command is a valid Command
Return: Status whether the command has reached the satellite or not

**get(payload: payloadIdentifier, parameter: parameterIdentifier): data**

SRFUR74

Must have

Retrieve the given parameter from the payload.
Pre: true
Return: The requested parameter value

**fetch(payload: payloadIdentifier, datum: dataIdentifier): data**

SRFUR75

Must have

Download the datum from the payload.
Pre: true
Return: The requested datum

**store(payload: payloadIdentifier, experiment: experimentInfo): status**
SRFUR76
Upload the experiment to the payload.

Pre: true
Return: Status whether the experiment has reached the satellite or not

erase(payload : payloadIdentifier, datum : dataIdentifier) : status

SRFUR77
Remove the given datum from the payload.

Pre: true
Return: Status whether the command has reached the satellite or not

list(payload : payloadIdentifier, path : pathIdentifier) : dataIdentifierList

SRFUR78
List the available data for the given path on the payload.

Pre: true
Return: The requested list

schedule(payload : payloadIdentifier, experiment : experimentIdentifier, startMoment : time) : status

SRFUR79
Schedule the experiment to start execution on the given moment on the payload.

Pre: true
Return: Status whether the command has reached the satellite or not

run(payload : payloadIdentifier, experiment : experimentIdentifier) : status

SRFUR80
Start the test on the payload.

Pre: true
Return: Status whether the command has reached the satellite or not

kill(payload : payloadIdentifier, experiment : experimentIdentifier) : status

SRFUR81
Stop the given test on the payload.

Pre: true
Return: Status whether the command has reached the satellite or not

3.1.22 MCC

SRFUR82
The MCC class provides communication with the MCCs.

Methods

requestBooking()

SRFUR83
Request a booking.

Pre: true
Return: Whether the booking has been made or not

getListOfOpportunities()
CHAPTER 3. SPECIFIC REQUIREMENTS

SRFUR84
Must have
Gets the list of possible opportunities to schedule an experiment
Pre: true
Return: A list of opportunities

3.1.23 Controller

Methods
createExperiment()

SRFUR85
Must have
Creates the experiment object.
Pre: true
Post: The experiment object is created

listOfCommands()

SRFUR86
Must have
Returns a list of the associated payload commands
Pre: true
Return: List of the associated payload commands

showThumbnails()

SRFUR87
Must have
Propagates to observation and shows the result on the screen
Pre: true
Post: The thumbnails are shown on the screen.

marked(t: Thumbnail)

SRFUR88
Must have
Marks a thumbnail for download
Pre: true
Post: Thumbnail has been marked for download.

observeExperiment()

SRFUR89
Must have
Controls all actions that relate to observing the experiment and shows the data
Pre: true
Post: Experiment can be seen on screen

Non-class model requirements
The following requirements are not from the class model, but are requirements from the tables, Petri nets, MSCs or the prototype.

SRFUR90
Must have
IMSETY shall only allow users to perform function they have rights to as showed in table 2.1

SRFUR91
Must have
IMSETY shall adhere to the login flow as seen in figure 2.3

SRFUR92
Must have
IMSETY shall adhere to the payload life cycle as shown in figure 2.4
### CHAPTER 3. SPECIFIC REQUIREMENTS

<table>
<thead>
<tr>
<th>SRFUR Code</th>
<th>Requirement</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRFUR93</td>
<td>IMSETY shall report the rejection of a booking according to MSC figures 2.13 and 2.14 and prototype figures A.2 and A.7</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR94</td>
<td>IMSETY shall allow an administrator to set up possible satellite commands through an interface as described in the prototype section A.7.3</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR95</td>
<td>IMSETY shall not allow scientists to type in commands directly as discussed in the prototype section A.4.2</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR96</td>
<td>IMSETY will represent a timeline of one-way and two-way communication opportunities as shown in the prototype section A.4.3</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR97</td>
<td>Scientist are able to observe experiments on their own payload as described in equation 2.1</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR98</td>
<td>Commands used in experiments need to be predefined on payloads as described in equation 2.2</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR99</td>
<td>Data can only be gathered during an experiment as described in equation 2.4</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR100</td>
<td>Commands can be sent to experiments that are already executing as seen in figure A.8 and MSC figure 2.17</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR101</td>
<td>Experiment data can be observed in real time while the experiment is being executed during a window of opportunity as seen in MSC figure 2.16</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR102</td>
<td>IMSETY shall communicate failure of satellite communication to the scientist, see section A.7.4</td>
<td>Must have</td>
</tr>
<tr>
<td>SRFUR103</td>
<td>IMSETY shall communicate failure of communication with GENSO to the scientist, see section A.4.3</td>
<td>Must have</td>
</tr>
</tbody>
</table>

#### 3.2 Extra-functional requirements

<table>
<thead>
<tr>
<th>SREFR Code</th>
<th>Requirement</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SREFR1</td>
<td>A demonstration experiment will be included.</td>
<td>Should have</td>
</tr>
<tr>
<td>SREFR2</td>
<td>A plug-in for postprocessing will be included.</td>
<td>Could have</td>
</tr>
<tr>
<td>SREFR3</td>
<td>A plug-in for preprocessing will be included.</td>
<td>Could have</td>
</tr>
<tr>
<td>SREFR4</td>
<td>The system shall interface with GENSO through an MCC for purposes of booking</td>
<td>Must have</td>
</tr>
<tr>
<td>SREFR5</td>
<td>The system shall interface with an MCS for communication with satellites</td>
<td>Must have</td>
</tr>
<tr>
<td>SREFR6</td>
<td>The protocol for interfacing with the MCS will be based on a list of commands provided by the customer</td>
<td>Must have</td>
</tr>
<tr>
<td>SREFR7</td>
<td>The interface shall be accessible on Windows 2000/XP/Vista, Mac OS X, Sun Solaris and Linux as described in 2.4</td>
<td>Must have</td>
</tr>
</tbody>
</table>
### CHAPTER 3. SPECIFIC REQUIREMENTS

<table>
<thead>
<tr>
<th><strong>SREFR8</strong></th>
<th><strong>Must have</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The backend shall run on a Linux server as described in 2.4</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 4

Requirements traceability matrices

This chapter contains two matrices that can be used to match the specific requirements described in this document with the specific requirements described in the URD [3]. Section 4.1 maps each user requirement to one or more software requirements while section 4.2 maps each software requirement to one or more user requirements.

4.1 UR to SR

In this section, in the case that one of the user requirements is mapped to both a specific requirement describing a Petri net and a specific requirement describing an MSC, the Petri net describes the actor point of view, whereas the MSC describes what actions IMSETY takes upon receiving a request from an actor.

<table>
<thead>
<tr>
<th>UR</th>
<th>SR</th>
</tr>
</thead>
<tbody>
<tr>
<td>URCAR1</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR2</td>
<td>SRFUR2</td>
</tr>
<tr>
<td>URCAR3</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR4</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR5</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR6</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR7</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR8</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR9</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR10</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR11</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR12</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR13</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR14</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR15</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR16</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR17</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR18</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR19</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR20</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR21</td>
<td>SRFUR1</td>
</tr>
<tr>
<td>URCAR22</td>
<td>SRFUR1</td>
</tr>
</tbody>
</table>
### CHAPTER 4. REQUIREMENTS TRACEABILITY MATRICES

<table>
<thead>
<tr>
<th>URCAR23</th>
<th>SRFUR15</th>
</tr>
</thead>
<tbody>
<tr>
<td>URCAR24</td>
<td>SRFUR16</td>
</tr>
<tr>
<td>URCAR25</td>
<td>SRFUR17</td>
</tr>
<tr>
<td>URCAR26</td>
<td>SRFUR18, SRFUR19</td>
</tr>
<tr>
<td>URCAR27</td>
<td>SRFUR20, SRFUR21, SRFUR22</td>
</tr>
<tr>
<td>URCAR28</td>
<td>SRFUR23, SRFUR24</td>
</tr>
<tr>
<td>URCAR29</td>
<td>SRFUR25, SRFUR26, SRFUR27</td>
</tr>
<tr>
<td>URCAR30</td>
<td>SRFUR28, SRFUR29, SRFUR30, SRFUR31</td>
</tr>
<tr>
<td>URCAR31</td>
<td>SRFUR32, SRFUR33, SRFUR34, SRFUR35</td>
</tr>
<tr>
<td>URCAR32</td>
<td>SRFUR36, SRFUR37, SRFUR38, SRFUR39</td>
</tr>
<tr>
<td>URCAR33</td>
<td>SRFUR40, SRFUR41</td>
</tr>
<tr>
<td>URCAR34</td>
<td>SRFUR42, SRFUR43, SRFUR44, SRFUR45</td>
</tr>
<tr>
<td>URCAR35</td>
<td>SRFUR46, SRFUR47, SRFUR48</td>
</tr>
<tr>
<td>URCAR36</td>
<td>SRFUR49, SRFUR50, SRFUR51</td>
</tr>
<tr>
<td>URCAR37</td>
<td>SRFUR52, SRFUR53, SRFUR54</td>
</tr>
<tr>
<td>URCAR38</td>
<td>SRFUR55, SRFUR56, SRFUR57</td>
</tr>
<tr>
<td>URCAR39</td>
<td>SRFUR58, SRFUR59, SRFUR60, SRFUR61</td>
</tr>
<tr>
<td>URCAR40</td>
<td>SRFUR62, SRFUR63, SRFUR64, SRFUR65</td>
</tr>
<tr>
<td>URCAR41</td>
<td>SRFUR66, SRFUR67, SRFUR68</td>
</tr>
<tr>
<td>URCAR42</td>
<td>SRFUR69, SRFUR70</td>
</tr>
<tr>
<td>URCAR43</td>
<td>SRFUR71, SRFUR72</td>
</tr>
<tr>
<td>URCAR44</td>
<td>SRFUR73, SRFUR74</td>
</tr>
<tr>
<td>URCAR45</td>
<td>SRFUR75, SRFUR76</td>
</tr>
<tr>
<td>URCAR46</td>
<td>SRFUR77, SRFUR78, SRFUR79</td>
</tr>
<tr>
<td>URCAR47</td>
<td>SRFUR80, SRFUR81</td>
</tr>
<tr>
<td>URCAR48</td>
<td>SRFUR82, SRFUR83</td>
</tr>
<tr>
<td>URCAR49</td>
<td>SRFUR84, SRFUR85</td>
</tr>
<tr>
<td>URCAR50</td>
<td>SRFUR86, SRFUR87</td>
</tr>
<tr>
<td>URCAR51</td>
<td>SRFUR88, SRFUR89</td>
</tr>
<tr>
<td>URCAR52</td>
<td>SRFUR90, SRFUR91</td>
</tr>
<tr>
<td>URCAR53</td>
<td>SRFUR92, SRFUR93</td>
</tr>
<tr>
<td>URCAR54</td>
<td>SRFUR94, SRFUR95</td>
</tr>
<tr>
<td>URCAR55</td>
<td>SRFUR96, SRFUR97</td>
</tr>
<tr>
<td>URCAR56</td>
<td>SRFUR98, SRFUR99</td>
</tr>
<tr>
<td>URCAR57</td>
<td>SRFUR100, SRFUR101</td>
</tr>
<tr>
<td>URCAR58</td>
<td>SRFUR102, SRFUR103</td>
</tr>
<tr>
<td>URCAR59</td>
<td>SRFUR104, SRFUR105</td>
</tr>
<tr>
<td>URCAR60</td>
<td>SRFUR106, SRFUR107</td>
</tr>
<tr>
<td>URCAR61</td>
<td>SRFUR108, SRFUR109</td>
</tr>
<tr>
<td>URCAR62</td>
<td>SRFUR110, SRFUR111</td>
</tr>
<tr>
<td>URCAR63</td>
<td>SRFUR112, SRFUR113</td>
</tr>
<tr>
<td>URCAR64</td>
<td>SRFUR114, SRFUR115</td>
</tr>
<tr>
<td>URCAR65</td>
<td>SRFUR116, SRFUR117</td>
</tr>
<tr>
<td>URCAR66</td>
<td>SRFUR118, SRFUR119</td>
</tr>
<tr>
<td>URCOR1</td>
<td>SRFUR120, SRFUR121</td>
</tr>
<tr>
<td>URCOR2</td>
<td>SRFUR122, SRFUR123</td>
</tr>
<tr>
<td>URCOR3</td>
<td>SRFUR124, SRFUR125, SRFUR126, SRFUR127</td>
</tr>
<tr>
<td>URCOR4</td>
<td>SRFUR128, SRFUR129</td>
</tr>
<tr>
<td>URCOR5</td>
<td>SRFUR130</td>
</tr>
</tbody>
</table>
### 4.2 SR to UR

<table>
<thead>
<tr>
<th>SR</th>
<th>UR</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRFUR1</td>
<td>URCAR1 URCAR3</td>
</tr>
<tr>
<td>SRFUR2</td>
<td>URCAR2</td>
</tr>
<tr>
<td>SRFUR3</td>
<td>URCAR1</td>
</tr>
<tr>
<td>SRFUR4</td>
<td>URCAR22</td>
</tr>
<tr>
<td>SRFUR5</td>
<td>URCAR23</td>
</tr>
<tr>
<td>SRFUR6</td>
<td>URCAR24</td>
</tr>
<tr>
<td>SRFUR7</td>
<td>URCAR25</td>
</tr>
<tr>
<td>SRFUR8</td>
<td>URCAR26</td>
</tr>
<tr>
<td>SRFUR9</td>
<td>URCAR26</td>
</tr>
<tr>
<td>SRFUR10</td>
<td>URCAR32</td>
</tr>
<tr>
<td>SRFUR11</td>
<td>URCAR32</td>
</tr>
<tr>
<td>SRFUR12</td>
<td>URCAR16</td>
</tr>
<tr>
<td>SRFUR13</td>
<td>URCAR11</td>
</tr>
<tr>
<td>SRFUR14</td>
<td>URCAR12</td>
</tr>
<tr>
<td>SRFUR15</td>
<td>URCAR13</td>
</tr>
<tr>
<td>SRFUR16</td>
<td>URCAR14</td>
</tr>
<tr>
<td>SRFUR17</td>
<td>URCAR15</td>
</tr>
<tr>
<td>SRFUR18</td>
<td>URCAR17</td>
</tr>
<tr>
<td>SRFUR19</td>
<td>URCAR18</td>
</tr>
<tr>
<td>SRFUR20</td>
<td>URCAR20</td>
</tr>
<tr>
<td>SRFUR21</td>
<td>URCAR21</td>
</tr>
<tr>
<td>SRFUR22</td>
<td>URCAR31</td>
</tr>
<tr>
<td>SRFUR23</td>
<td>URCAR27</td>
</tr>
<tr>
<td>SRFUR24</td>
<td>URCAR27</td>
</tr>
<tr>
<td>SRFUR25</td>
<td>URCAR31</td>
</tr>
<tr>
<td>SRFUR26</td>
<td>URCAR30</td>
</tr>
<tr>
<td>SRFUR27</td>
<td>URCAR30</td>
</tr>
<tr>
<td>SRFUR28</td>
<td>URCAR30 URCAR32</td>
</tr>
<tr>
<td>SRFUR29</td>
<td>URCAR30</td>
</tr>
<tr>
<td>SRFUR30</td>
<td>URCAR32</td>
</tr>
<tr>
<td>SRFUR31</td>
<td>URCAR34</td>
</tr>
<tr>
<td>SRFUR32</td>
<td>URCAR35</td>
</tr>
<tr>
<td>SRFUR33</td>
<td>URCAR32</td>
</tr>
<tr>
<td>SRFUR34</td>
<td>URCAR32 URCAR36</td>
</tr>
<tr>
<td>SRFUR35</td>
<td>URCAR39</td>
</tr>
<tr>
<td>SRFUR36</td>
<td>URCAR39</td>
</tr>
<tr>
<td>SRFUR37</td>
<td>URCAR39</td>
</tr>
<tr>
<td>SRFUR38</td>
<td>URCAR39</td>
</tr>
<tr>
<td>SRFUR39</td>
<td>URCAR59 URCAR60</td>
</tr>
<tr>
<td>SRFUR40</td>
<td>URCAR59 URCAR60</td>
</tr>
<tr>
<td>SRFUR41</td>
<td>URCAR59 URCAR60</td>
</tr>
<tr>
<td>SRFUR42</td>
<td>URCAR41</td>
</tr>
<tr>
<td>SRFUR43</td>
<td>URCAR41</td>
</tr>
<tr>
<td>SRFUR44</td>
<td>URCAR41</td>
</tr>
<tr>
<td>SRFUR45</td>
<td>URCAR49</td>
</tr>
<tr>
<td>SRFUR46</td>
<td>URCAR41</td>
</tr>
<tr>
<td>SRFUR47</td>
<td>URCAR55</td>
</tr>
<tr>
<td>SRFUR48</td>
<td>URCAR50</td>
</tr>
<tr>
<td>SRFUR49</td>
<td>URCAR51</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>SRFUR50</td>
<td>URCAR57</td>
</tr>
<tr>
<td>SRFUR51</td>
<td>URCAR40</td>
</tr>
<tr>
<td>SRFUR52</td>
<td>URCAR41</td>
</tr>
<tr>
<td>SRFUR53</td>
<td>URCAR45</td>
</tr>
<tr>
<td>SRFUR54</td>
<td>URCAR29</td>
</tr>
<tr>
<td>SRFUR55</td>
<td>URCAR52</td>
</tr>
<tr>
<td>SRFUR56</td>
<td>URCAR46</td>
</tr>
<tr>
<td>SRFUR57</td>
<td>URCAR45</td>
</tr>
<tr>
<td>SRFUR58</td>
<td>URCAR43</td>
</tr>
<tr>
<td>SRFUR59</td>
<td>URCAR55</td>
</tr>
<tr>
<td>SRFUR60</td>
<td>URCAR51</td>
</tr>
<tr>
<td>SRFUR61</td>
<td>URCAR43</td>
</tr>
<tr>
<td>SRFUR62</td>
<td>URCAR44</td>
</tr>
<tr>
<td>SRFUR63</td>
<td>URCAR43</td>
</tr>
<tr>
<td>SRFUR64</td>
<td>URCAR45</td>
</tr>
<tr>
<td>SRFUR65</td>
<td>URCAR45</td>
</tr>
<tr>
<td>SRFUR66</td>
<td>URCAR45</td>
</tr>
<tr>
<td>SRFUR67</td>
<td>URCAR41</td>
</tr>
<tr>
<td>SRFUR68</td>
<td>URCAR55</td>
</tr>
<tr>
<td>SRFUR69</td>
<td>URCAR64</td>
</tr>
<tr>
<td>SRFUR70</td>
<td>URCAR10</td>
</tr>
<tr>
<td>SRFUR71</td>
<td>URCAR64</td>
</tr>
<tr>
<td>SRFUR72</td>
<td>URCOR2</td>
</tr>
<tr>
<td>SRFUR73</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR74</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR75</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR76</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR77</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR78</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR79</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR80</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR81</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SRFUR82</td>
<td>URCOR1</td>
</tr>
<tr>
<td>SRFUR83</td>
<td>URCAR59</td>
</tr>
<tr>
<td>SRFUR84</td>
<td>URCAR34</td>
</tr>
<tr>
<td>SRFUR85</td>
<td>URCAR32</td>
</tr>
<tr>
<td>SRFUR86</td>
<td>URCAR31</td>
</tr>
<tr>
<td>SRFUR87</td>
<td>URCAR55</td>
</tr>
<tr>
<td>SRFUR88</td>
<td>URCAR56</td>
</tr>
<tr>
<td>SRFUR89</td>
<td>URCAR41</td>
</tr>
<tr>
<td>SRFUR90</td>
<td>URCAR7</td>
</tr>
<tr>
<td>SRFUR91</td>
<td>URCAR4</td>
</tr>
<tr>
<td>SRFUR92</td>
<td>URCAR19</td>
</tr>
<tr>
<td>SRFUR93</td>
<td>URCAR63</td>
</tr>
<tr>
<td>SRFUR94</td>
<td>URCAR28</td>
</tr>
<tr>
<td>SRFUR95</td>
<td>URCAR33</td>
</tr>
<tr>
<td>SRFUR96</td>
<td>URCAR58</td>
</tr>
<tr>
<td>SRFUR97</td>
<td>URCAR9</td>
</tr>
<tr>
<td>SRFUR98</td>
<td>URCAR27</td>
</tr>
<tr>
<td>SRFUR99</td>
<td>URCAR48</td>
</tr>
<tr>
<td>SRFUR100</td>
<td>URCAR37</td>
</tr>
<tr>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>SRFUR101</td>
<td>URCAR42</td>
</tr>
<tr>
<td>SRFUR102</td>
<td>URCAR61</td>
</tr>
<tr>
<td>SRFUR103</td>
<td>URCAR62</td>
</tr>
<tr>
<td>SREFR1</td>
<td>URCAR6</td>
</tr>
<tr>
<td>SREFR2</td>
<td>URCAR54</td>
</tr>
<tr>
<td>SREFR3</td>
<td>URCAR53</td>
</tr>
<tr>
<td>SREFR4</td>
<td>URCOR1</td>
</tr>
<tr>
<td>SREFR5</td>
<td>URCOR2</td>
</tr>
<tr>
<td>SREFR6</td>
<td>URCOR3</td>
</tr>
<tr>
<td>SREFR7</td>
<td>URCOR4</td>
</tr>
<tr>
<td>SREFR8</td>
<td>URCOR5</td>
</tr>
</tbody>
</table>
Appendix A

Prototype description

This chapter contains a short description of the different views in the prototype. Most sections in which a view is described will also contain a screenshot to clarify the description. Note that “he” and “his” may also be read as “she” and “her”.

Figure A.1 depicts the order in which views can be opened. An arrow from view A to view B means that the user can choose to open the latter if he is currently confronted with view A. Consequently, an arrow from view A to view B and vice versa means that the user can not only open view B from view A, but also view A from view B. It is important to note that in case a user closes a view, he will return to the view which brought him there.

![Diagram of view order]

Figure A.1: The order in which views can be opened

A.1 Rights

Only one application will be built, which can be used by all three user groups, i.e. the administrators, the scientists and the observers. In table A.1 the rights for all views are given. When a certain user group has no rights to open a certain view, this view will be invisible for that user. "✓" Says that only the experiments are shown to which the user has access.

In the rest of the document we will pay no attention to access rights on views, we will just use the word user instead of for example administrator or observer.
APPENDIX A. PROTOTYPE DESCRIPTION

<table>
<thead>
<tr>
<th>Views</th>
<th>Users</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Administrator</td>
</tr>
<tr>
<td>Main view</td>
<td>✓</td>
</tr>
<tr>
<td>Login</td>
<td>✓</td>
</tr>
<tr>
<td>Preferences</td>
<td>✓</td>
</tr>
<tr>
<td>Create an experiment</td>
<td>✓</td>
</tr>
<tr>
<td>Edit experiment</td>
<td>✓</td>
</tr>
<tr>
<td>Schedule experiment</td>
<td>✓</td>
</tr>
<tr>
<td>Observation and</td>
<td>✓</td>
</tr>
<tr>
<td>intervention</td>
<td>✓</td>
</tr>
<tr>
<td>Observation data</td>
<td>✓</td>
</tr>
<tr>
<td>Manage observers</td>
<td>✓</td>
</tr>
<tr>
<td>Manage accounts</td>
<td>✓</td>
</tr>
<tr>
<td>Manage satellites</td>
<td>✓</td>
</tr>
<tr>
<td>Manage payloads</td>
<td>✓</td>
</tr>
<tr>
<td>Communication logs</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table A.1: User rights to views

A.2 Main view

The Main view (see figure A.2) is the first view which is presented to the user when he opens the client. On top of this view will be the Login view (see section A.3.1), which prohibits the user to access the Main view as long he does not log in. The Main view of the client consists of a list of experiments grouped by satellite and then by payload. It should be noted that the user only can see those entities to which he has access. A payload can be folded or unfolded, to toggle the display of its experiments. In this view, the user will also be informed about the status of each experiment.

![Figure A.2: The Main view](image-url)
A.3 File

A.3.1 Login

The Login view (see figure A.3) is a simple login dialog with an extra button, the Server button. When this button is pressed, it will invoke a new window in which the server the client connects to can be configured. This client-server architecture will be described in the Architectural Design Document. During the login phase, the user is informed about the login progress by simple messages in the Login view.

![Figure A.3: The Login view](image)

A.3.2 Preferences

In the Preferences view (see figure A.4) the user can edit his preferences. This includes the option to change the server as in the Login view (see section A.3.1). On the Preferences view the user can also, for example, change his password.

![Figure A.4: The Preferences view](image)

A.4 Experiment

A user is always able to add an experiment, but he can only edit or remove an experiment once he selected it from the experiment list on the Main view (see section A.2). The easiest action is to delete an experiment. A user can delete an experiment by clicking on a certain experiment in the experiment list and selecting Remove from the Experiment menu. All experiment-specific views
A.4.1 Create an experiment

In the Create an experiment view (see figure A.5), the user is able to create new experiments. In this view he needs to fill out some details about the experiment, for example its name and the payload it belongs to. When the user has filled out those details he has to press the OK button to proceed, or the Cancel button to stop. This will take him to the editing part of the Experiment view (see section A.4.2), in which the test sequence will initially be empty.

![Figure A.5: The Create an experiment view](image)

A.4.2 Edit experiment

In the editing part of the Experiment view (see figure A.6), the user can schedule commands, with certain parameters, in a test sequence. He can do this by selecting a command in the Commands part of the Command box, editing its parameters in the Parameters part and choosing a relative time with respect to the start of the experiment. After these are set, the user has to press the Add button to add this command to the test sequence. The test sequence contains a list of commands, sorted by time. To edit a certain command the user has to click on that specific command and press the Edit button. He is now able to edit the command in the Command box. If the user is satisfied with the changes, he can press the Update button, the Cancel button if he is not.

![Figure A.6: The editing part of the Experiment view](image)
Possible commands include enabling and disabling built-in cameras. When a camera is enabled and there is at least one-way communication possible with the satellite during the execution of an experiment, users are able to see real-time images captured by the camera. Because these images are not saved in high-resolution by default, additional commands for enabling and disabling the capturing of high-resolution data are available.

It is important for the user to add the \textit{Finish experiment} command to the test sequence in order to let the satellite know when the experiment is ended. To remind the user of this, a red text will be displayed on the view until he adds this command. Once the user is satisfied with the test sequence he has made, he can press the \textit{Schedule} button to go to the scheduling part of the \textit{Experiment} view (see section \ref{sec:scheduling}).

### A.4.3 Schedule experiment

In the scheduling part of the \textit{Experiment} view (see figure \ref{fig:schedule}), the user can schedule an experiment. On the left part of this view, the user will see the test sequence of the experiment he wants to schedule. On the right part of the view, the part where the actual scheduling takes place, the user is presented with a list of passes combined with a timeline initially showing all passes for a particular date. Each pass can be checked or unchecked to toggle the display of the pass in the timeline. When a particular pass is selected, the timeline will automatically scroll to the time interval that pass is in. Passes are represented in the timeline as blue rectangles in which a ‘I’ is drawn when there is a possibility for one-way communication and a ‘II’ when there is possibility for two-way communication. On the right in this timeline, a user will see all other scheduled experiments on the payload, which are drawn as red rectangles. Note that these red rectangles inform the scientist of periods in which he can not schedule his own experiment, because of the constraint that per satellite there is at most one experiment executing at a time. To schedule his experiment, the user can scroll his experiment (the green rectangle in the middle) through the timeline or set the start time in an input field below it. The period of the experiment during which a camera is enabled will be coloured light green to help picking a convenient start time.

The scheduling part of the \textit{Experiment} view also contains a list of scheduled experiments, which informs the user of periods in which he can not schedule his experiment. The user is also informed about the current status of the scheduled experiment, for example if the passes have already been booked or if the booking is rejected.

![Figure A.7: The scheduling part of the Experiment view](image-url)
A.5 Observe experiment

In the Observation and intervention view (see figure A.8), the user can observe at most two experiments at a time. The default view consists of two similar components, one for each experiment. Each component consists of a postprocessing component and a visualization component on the upper part of the view, and a component for displaying logs on the bottom part.

![Observation and intervention view](image)

Figure A.8: The Observation and intervention view

For each experiment observed in this view, a visualization and a postprocessing part is presented. The first visualizes the observation data if this is possible. The latter enables the user to apply some filters for postprocessing the visualization. This can be done easily by clicking the button labelled with a plus sign, which will invoke a new view with an overview of all available filters. In this view a filter can be selected and its parameters can be set. Once a filter is added, the parameters for this plug-in can be changed in the options part of the postprocessing component. The order of the filters can be changed by pressing the up and down arrows right to the filter list, and filters can be enabled or disabled by ticking or un-ticking the checkbox in front of it.

To intervene with an experiment, the commands component needs to be invoked. This can be done by selecting Send commands from the View menu. Because only one user can intervene with an experiment at a time, the commands component will only be invoked when there is no other user intervening with that experiment. The commands component consists of the same command box as in the editing part of the Experiment view (see section A.4.2) plus two checkboxes, one for each experiment. If the user wants to send a command to a payload belonging to a certain experiment, he needs to specify with which experiment he wants to intervene. He can do this by ticking the appropriate checkbox above the commands component. This will load the commands belonging to that specific payload. When both checkboxes are checked, the intersection of the command sets is presented in the command list. When the user sees something that might interest him, he can press the button labelled with a star-icon. When this button is pressed, all images from that point on are marked as interesting.
APPENDIX A. PROTOTYPE DESCRIPTION

until the button gets pressed again.

A.6 Observation data

In the Observation data view (see figure A.9), users can view observation data which is gathered during (partially) executed experiments. This view is divided into two tabs. On the first tab the user can filter the images on a few criteria, for example on payload, experiment and observation. Once he has set the filter criteria, he could press the Filter button to retrieve the thumbnails matching the criteria. All images marked as interesting during observations (see section A.5) will be displayed with a star-icon on it. To select all these images from the current (possibly filtered) set of thumbnails, the user can press the Select interesting button. There are three possible actions on a (set of) thumbnail(s), which can be invoked by selecting the appropriate action in the context menu.

A thumbnail can be:

- queued for downloading in high-resolution from the satellite, if available
- exported locally after applying postprocessors
- added locally to the batch postprocessing list
- queued for deletion from the satellite

Each thumbnail has a coloured border around it to represent its state. Blue represents a normal thumbnail, red represents an image that is queued for downloading in high-resolution and green represents that the thumbnail is also available in a high-resolution version. When the user selects the option to queue an image for downloading in high-resolution, this item will be added to the queue list on the second tab.

![Figure A.9: The Observation data view](image)

If the user chooses to edit an image, a new view will be opened. In this view a postprocessing component, as used in A.5, accompanies the visualization of the image. Images can also be postprocessed in a batch, which makes it easy to process multiple images at once. In an apart batch postprocessing/exporting component on the view, a list is represented which represents thumbnails selected for batch postprocessing/exporting and a postprocessing component as in A.5 is represented. It also contains export functionalities, which enables the user to export data to other file formats.
A.6.1 Manage observers

In the Manage observers view (see figure A.10) users can manage observers per experiment. This can either be to grant someone the right to be an observer on that particular experiment or to take away that right. The Manage observers view can be accessed in two different ways. One way is to click on the Observers button in the Experiment view (see section A.4.2 and section A.4.3). The second, slightly shorter, way is to select an experiment in the experiment list on the Main view (see section A.2) and select Manage observers from the Experiment menu. Both ways will take the user to the same view, the Manage observers view.

![Manage observers view](image)

On the upper part of this view is a small details section, in which some details of the experiment and its status are presented to the user. The other part of the view is filled with a component in which the user can grant or take away someone’s right to be an observer. Granting observer rights can be done easily, either by selecting the appropriate names in the list on the left and clicking the triangle pointing to the right or by dragging the selected names into the right list. Taking away observer rights goes in the opposite way of granting observer rights. Changes in observer rights will be applied when the user clicks on the OK button or will be dismissed in case the user clicks on the Cancel button.

A.7 Administration

A.7.1 Manage accounts

In the Manage accounts view (see figure A.11) users can add other accounts to the system or edit details of existing accounts. The view consists of a list of all accounts on the left and their properties on the right. To edit the details of an account, the user has to select the specific accounts and press the Edit button. This will invoke a new view in which all details of the accounts can be edited.

When the user is satisfied with the changes, he could press the OK button to apply the changes. In case the user is not confident with the changes, he could press the Cancel button, so that no changes will be applied to the account. Adding an accounts is analogous to editing an existing accounts, i.e. when the Add button is pressed there will be a new view presented in which all details of the accounts can be filled in. To remove an account, the user has to select the account and press the Remove button. Both the Edit and Remove actions are also available in the context menu.
A.7.2 Manage satellites

The Manage satellites view is similar to the Manage accounts view (see section A.7.1) in the way that adding a new satellite, editing an existing satellite and removing a satellite requires the same sort of actions.

A.7.3 Manage payloads

In the Manage payloads view users can manage payloads. That is add, edit and delete payloads. This view is also similar to the Manage accounts view (see section A.7.1), because the Add, Edit and Remove buttons behave in the same way. There is only one difference, namely the properties view which will be shown when Add or Edit is pressed. This view is divided into multiple tabs, i.e. the General, Users and Commands tab. In the General tab, general info of the satellite can be edited, e.g. the MCS to which IMSETY will connect. The Users tab contains a component in which scientists can be coupled to a payload. In the Commands tab (see figure A.12) there is a component in which the commands for that payload can be specified and there is also the possibility to import and export these commands.

A.7.4 Communication logs

In the Communication logs view there is a box in which the user can select the date of the logs he wants to see. But the user can also make the use of other filter criteria. Once the user has set the search criteria, he should press the Filter button in order to request the logs. These logs will
be displayed in the lower half of the view. Examples of logs the user can retrieve on this view, are failure logs of satellite communication and failure logs of communication with GENSO.

![Communication logs view](image)

Figure A.13: The *Communication logs* view
Appendix B

External interfaces

Figure B.1: External interfaces diagram

IMSETY communicates through several external interfaces. For each satellite there is an MCC and an MCS as depicted in figure B.1. We will describe the services provided by these systems and the minimal interfaces needed for the communication. The protocol that will be used for the communication with these systems will be given in the Interface and Protocol Definition (IPD) [2].

B.1 The GENSO Mission Control Client

The GENSO network which provides access to the satellite. The MCC is our interface to the GENSO network. In order to use the GENSO network for satellite communication, ground station access needs to be booked. For booking we use the interface provided for this purpose by the MCC.

The communication with the MCC will be asynchronous. This is needed as it may take time before the GENSO negotiation mechanism has made a decision whether a requested booking will be confirmed or rejected. The functions listed below describe the interface the MCC component of IMSETY will provide to the other components. The arguments of the listed functions are an abstract form of the messages that will be sent to and/or received from the MCC. An exact definition of these messages will be formulated along with the protocol.

The abstract bookingInfo type will contain all information to identify a booking and to construct the message that will be sent to the MCC. A bookingList will contain a number of bookings. Because communication is asynchronous, the systems MCC component returns an id when
a method is called. This id is used to identify the answer from the MCC.

B.1.1 Booking

requestBooking(booking : bookingInfo) : integer

Request the booking of a pass. Possible responses:

- booked(id : integer, booking : bookingInfo) : void,
  the booking has been confirmed by the MCC.

- rejected(id : integer) : void,
  the booking has been rejected by the MCC.

- failure(id : integer) : void,
  the MCC failed to handle the booking request.

getListOfOppurtunities(period : (time, time)) : integer

Request a list of possible bookings. Possible responses:

- listOfOppurtunities(id : integer, list : bookingList) : void,
  the list of possible bookings during the given period.

- failure(id : integer) : void,
  the MCC failed to handle the list request.

B.2 The satellite’s Mission Control Software

The MCS is our interface to the satellite. Once a connection to the satellite has been established
commands can be sent. This interface can be used by scientists to manipulate their experiments.

The MCS also provides possibilities to manage the satellite, these commands will be hidden for
the scientist, but will be used by IMSETY for satellite data maintenance.

Communication with the MCS will be synchronous, (almost) immediately returning a status
value or some kind of data. The functions listed below describe the interface the MCS component
of IMSETY will provide to the other components. The arguments of the listed functions are an
abstract form of the command parameters that we will send to the MCS. An exact definition of
these parameters will be formulated along with the protocol.

In this interface description abstract data types are used. The status indicates whether the
command reached the satellite or not. The received data may be booleans, integers, floats, or
large binary objects. The abstract experimentInfo type will contain all information about an
experiment that the satellite needs, such as the list of commands.

B.2.1 Experiment / Payload specific commands

exec(payload : payloadIdentifier, command : string) : status

Execute the given command (will be parsed by the MCS).

B.2.2 Data handling (communication & management)

get(payload : payloadIdentifier, parameter : parameterIdentifier) : data

Retrieve a parameter value (or measurement) from the satellite.
fetch(payload : payloadIdentifier, datum : dataIdentifier) : data
Download a datum from the satellite. For example fetch the high-resolution image belonging to a thumbnail.

store(payload : payloadIdentifier, experiment : experimentInfo) : status
Upload an experiment (a series of scheduled commands) to the satellite.

erase(payload : payloadIdentifier, datum : dataIdentifier) : status
Remove a datum from the satellite. For example remove a thumbnail or a high-resolution image.

list(payload : payloadIdentifier, path : pathIdentifier) : dataIdentifierList
List the available data for the given path.

schedule(payload : payloadIdentifier, experiment : experimentIdentifier, startMoment : time) : status
Schedule an experiment to start execution on the given moment.

run(payload : payloadIdentifier, experiment : experimentIdentifier) : status
Start a test which has been stored earlier.

kill(payload : payloadIdentifier, experiment : experimentIdentifier) : status
Stop a test which is currently running.