Development and Implementation of a Mission Planning Tool for SONATE

Thomas Rapp

Space Engineering, master's level (120 credits)
2017

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering
Master Thesis

Development and Implementation of a Mission Planning Tool for SONATE

submitted by

Thomas Rapp

to the Department of Mathematics and Computer Science
in Partial Fulfilment of the Requirements of the Degree of Master of Science
with the Major of Space Science and Technology

Co-funded by the Erasmus+ Programme of the European Union
Master Thesis

Development and Implementation of a Mission Planning Tool for SONATE

submitted by

Thomas Rapp

born on 15th February 1992 in Munich

to the Department of Mathematics and Computer Science
in Partial Fulfilment of the Requirements of the Degree of Master of Science
with the Major of Space Science and Technology

Created at:
Chair of Aerospace Information Technology
Professorship of Space Technology
Julius–Maximilians–University of Würzburg

Supervisor:
M.Sc. Tobias Schwarz

Examiners:
Prof. Dr.-Ing. Hakan Kayal (JMUW)
Dr. Leonard Felicetti (LTU)

Submission Date:
1st August 2017

1thomas.rapp@stud-mail.uni-wuerzburg.de
Declaration of Authorship

I declare that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university. Formulations and ideas taken from other sources are cited as such. This work has not been published before.

Würzburg, 1st August 2017

(Thomas Rapp)
This page is intentionally left blank.
ACKNOWLEDGEMENTS

I would like to thank M.Sc. Tobias Schwarz, Prof. Dr. Hakan Kayal and Dr. Leonard Felicetti for their supervision of this thesis and their advice in many different aspects.

Furthermore, I would like to thank my parents Gabriele and Michael Rapp for their outstanding support throughout my studies, including stays abroad, the excursion to a second Bachelor’s degree, and various other projects.

Last but not least, I would like to thank Lars Vogel, Thomas Schindl and greg-449, who are members of and contributors to the Eclipse community. Although I have never had direct contact to them, they seem somehow familiar thanks to their numerous and excellent tutorials, forum discussions, and advice on the Eclipse Rich Client Platform, which helped me a lot during the implementation process of this thesis project.
This page is intentionally left blank.
In the scope of the master’s project which is documented with the present thesis a mission planning tool (MPT) for SONATE was developed and implemented. After a thorough research on the current state of the art of MPTs and taking especially the early stage of the SONATE mission into account, it was decided to develop a generic timeline-based MPT. In contrast to existing MPTs a system is envisioned which is both powerful, regarding advanced features like resource control, and applicable for small satellite missions regarding the overall complexity and the associated configuration and training effort. Although it was obvious from an early stage that this vision cannot be reached in the scope of this project, it was kept during the project definition, object oriented analysis and early design stages in order to allow future extensions. Also the decision to develop the MPT on top of the Eclipse Rich Client Platform is mainly due to the argument of future extensibility.

The MPT, which is released with this thesis, hence is a very basic generic timeline-based MPT omitting all possible advanced features like resource control or procedure validation, but featuring all essential parts of a MPT, i.e. modelling of procedures, scheduling of activities, and the generation of telecommand sequences. Furthermore, the user is supported by an intuitive graphical user interface. The thesis documents the development process, thus giving a broad understanding of the design and the implementation. For specific details of the implementation one may also refer to the separate technical documentation, while a user handbook included as appendix.

The characteristics of the SONATE mission as a technology demonstrator for highly autonomous systems raise several important questions regarding the overall mission planning process. Therefore, besides the actual development of the MPT, those questions are discussed in a theoretical manner in the scope of this thesis, taking also account of the general emergence of highly autonomous satellites systems. Three concepts, Safe Planning, Sigma Resource Propagation, and Direct Telemetry Feedback, are proposed to face the challenges rising from the foreseen alternation of phases of classical mission operations and phases of autonomous operations of the satellite.

Concluding the thesis, the final software product’s features and capabilities are verified against the previously defined requirements and thus the overall success of the project is determined to be a 100% success fulfilling all primary project objectives. Finally, several fields for further research on the topic in general and work on the MPT itself are identified and outlined to pave the way for follow-up projects.
This page is intentionally left blank.
CHAPTER 1 – INTRODUCTION

CHAPTER 2 – SURVEY OF RELATED RESEARCH

CHAPTER 3 – PROJECT DEFINITION
  3.1 Project Statement
  3.2 Project Objectives
    3.2.1 Primary Objectives
    3.2.2 Secondary Objectives
  3.3 Constraints
  3.4 Requirements

CHAPTER 4 – OBJECT ORIENTED ANALYSIS
  4.1 System Use Cases
    4.1.1 System Context
    4.1.2 Use Cases of the Systems Engineer
    4.1.3 Use Cases of the Planning Engineer
    4.1.4 Use Cases of the Operator
    4.1.5 General Remarks on the Use Cases
  4.2 Subject Specific Classes
  4.3 Definition of System Interfaces
    4.3.1 Data Interfaces
      Telecommand Database
      Flight Dynamics
      Telecommand Client
    4.3.2 Graphical User Interface
      Planning Perspective
      Modelling Perspective
      Monitoring Perspective
    4.3.3 Other User Interfaces

CHAPTER 5 – NOVEL CONCEPTS FOR MISSION PLANNING TOOLS FOR AUTONOMOUS SATELLITES
  5.1 General Problem
  5.2 Safe Planning
  5.3 Sigma Resource Propagation
### CHAPTER 8 – VERIFICATION AND TESTING 63

- **8.1 Test Philosophy** 63
- **8.2 Verification** 65
  - **8.2.1 General Remarks** 65
  - **8.2.2 Verification of Operational Requirements** 65
  - **8.2.3 Verification of Design Requirements** 66
  - **8.2.4 Verification of Performance Requirements** 67
  - **8.2.5 Verification of Functional Requirements** 67
  - **8.2.6 Verification of the Project Objectives & Project Statements** 69

### CHAPTER 9 – FURTHER WORK 71

### CHAPTER 10 – CONCLUSION 73

### APPENDIX A – REQUIREMENTS 75

- **A.1 Functional Requirements** 75
- **A.2 Performance Requirements** 83
- **A.3 Design Requirements** 86
- **A.4 Operational Requirements** 88

### APPENDIX B – USER HANDBOOK 89

- **B.1 Overview** 89
- **B.2 Installation & First Execution** 90
- **B.3 Preferences** 90
- **B.4 Procedure Editing Perspective** 96
- **B.5 Planning Perspective** 100
- **B.6 SoE Monitor Perspective** 108

### APPENDIX C – INDEX OF SOURCE DVD 111

### REGISTERS 113

- **List of Figures** 113
- **List of Tables** 114
- **List of Listings** 114

### REFERENCES 115
This page is intentionally left blank.
CHAPTER 1

Introduction

Failing to plan is planning to fail.

Alan Lakein

This modern idiom, which is commonly attributed to Alan Lakein and in different variants also to Sir Winston Churchill and Benjamin Franklin, targets at personal time management or project management in general, but definitely also holds true for the operations of each and every space mission regardless of its scale, lifespan, or objectives. Incomplete or faulty planning of the activities of the remote controlled satellite puts severe threats onto the satellite’s “life” and the overall mission success. While most people still use simple calendars and todo-lists - even if it is a smartphone app - to schedule their daily routines, the planning for space missions is much more complex. Therefore, since the beginning of space flight, dedicated software tools to coordinate and plan the activities of a space mission, so called mission planning tools (MPT) were developed and supported the mission operations teams.

Also the SONATE mission, a 3U-picosatellite mission currently under development at the Professorship of Space Technology of Julius-Maximilians University of Würzburg, shall be supported by such a mission planning tool. Although there are several commercial MPTs available, it was expected that due to the limited budget and the special requirements of SONATE, it is more efficient to develop an own solution from scratch than to adapt and customize commercial tools.

Therefore, a MPT for SONATE was to be designed and implemented in the scope of this master’s project which is conducted to the partial fulfillment of the requirements for the degree Master of Space Science and Technology at the Julius-Maximilians University of Würzburg and Luleå Tekniska Universitetet. Despite the decision to develop an own solution, thorough research of related work and existing state of the art concepts for MPTs was the first part of this project and is summarized in chapter 2.

The third chapter defines the objectives and the actual scope of this thesis project. Already during the project definition, and thus throughout the whole de-
velopment, especially three unique aspects of the SONATE mission were regarded. Firstly, SONATE is a technology demonstration mission for highly autonomous satellites. This rises several questions regarding how the mission planning process is affected when the satellite itself takes over control of the conducted activities from the mission control center on ground during autonomous phases. Those questions are so far not broadly discussed in literature. Therefore, they are discussed theoretically and abstracted from the specific mission in chapter 5 and concepts to face this new situation are developed. As they were risen in the context of SONATE, the concepts are of course also applicable for the mission even though theoretically discussed here. In spite of the fact the better part of this thesis deals with the development of the MPT, which is probably only of limited relevance for the scientific community, the development of those concepts is regarded as profound contribution to the scientific community of space mission planning in the light of the emergence of autonomous satellites and thus justifies the classification of this project as a master’s project.

Secondly, both the desired capability to scale as many subsystems of SONATE as possible to a follow-up 27U-nanosat mission as well as the early stage of the mission, render a generic design and implementation of the MPT preferable. As by now many details of the mission which are important for the mission planning, like the set of available telecommands, are not yet precisely specified, it is favorable to separate the implementation of the functionalities from the model of the actual mission. While the modelling itself is only partly done in the scope of this project to demonstrate the functions of the MPT and the model could be modified throughout the whole mission, the core functionalities and algorithms of the MPT were implemented and are ready to use. Also from a point of view of software design, this is favorable and the MPT is easier to maintain, adapt for future missions, i.e. the envisioned 27U-nanosat mission or perhaps completely other ones, or even extend by more advanced features.

Based on the definition, the use cases, the subject specific classes and system interfaces are assessed in chapter 4 in a standardized way, the so called object oriented analysis. Both the analysis and the novel concepts for MPTs for autonomous satellites in mind, one passes on to the actual design of the MPT which is discussed in chapter 6. Chapter 7 summarizes the implementation process and provides remarks on the implementation of the MPT which are of special interest for future developers.

Finally, as the last part of this master’s project, the software product was thoroughly tested, evaluated and verified against its requirements. While chapter 8 describes the test philosophy and presents the results of the verification, chapters 9 and 10 round out the thesis with an outlook on possible further work and concluding remarks.
CHAPTER 2

Survey of Related Research

From the very first days of spaceflight on, mission operations and especially mission planning was a crucial factor deciding between success and failure of a mission. Due to the complexity of the so-called planning problem, to assemble and coordinate a valid mission plan which puts as little risk on the mission as possible while ensuring as much science return as possible, quite soon software systems were developed to assist the planning engineers.

At first, those tools were custom tools developed from scratch for each and every mission according its specific requirements (cf. [Fukunaga et al., 1997]). But as space missions grew in number and complexity, as well as information technology matured, in the late 1990s the engineers in charge realized that it would significantly reduce development effort and the risk on the mission due to malfunctions of the MPT, if the development of the software core were abstracted from the model of a specific space mission. Thus, the big agencies started to develop modelling languages, which enable operational staff without software engineering background to model the mission, and corresponding software frameworks, which include the generic planning algorithms and the generic user interface and which are developed and maintained by software engineering teams (cf. [Smith et al., 1998] and [Lenzen, 2015, p. 171]). By now, those frameworks of course have grown in capabilities and complexity and contain different tools for different steps and use cases of mission planning and even fully commercial of the shelf systems have emerged [Gutierrez et al., 2005]. The capabilities of those tools include features like automated and fully autonomous scheduling to maximize efficiency, support of multi-satellite missions, support of short notice science requests, support of flexible ground station networks, web-based services to monitor the current mission plan from anywhere, support of collaboration of multiple planning engineers, automated compilation and upload of telecommand lists, and various more similarly advanced functionalities (cf. [Maurer et al., 2008] [Chien et al., 2000] [Kavelaars et al., 2009] [Wörle et al., 2016]). Nevertheless, also recent publications, like [Wörle et al., 2016], show that those frameworks are by no means "complete", but have to be regularly extended with plug-ins, work-arounds or even new tools, to meet the special
needs of new missions. It is needless to say that with increasing capabilities of the frameworks also the complexity of the configuration and modelling of the mission have grown, which requires profound training of the mission planning teams.

Despite their origin in different agencies and developed with focus on different factors, those tools share a lot of commonalities. They are classified as so called "timeline-based" mission planning systems (cf. [Chien et al., 2012]) for the following reasons. They all apply the same approach, which is probably the most intuitive one, to represent the whole mission plan, all activities and resources, on several timelines, which show the projected future as well as reported past course over time. The mission is basically modeled by resources (finite states, infinite states, depletable, non-depletable), constraints (time dependencies, functional dependencies, limits of resources, allocation of resources by an activity), and activities which are scheduled using "linear, possibly grounded timepoints" [Chien et al., 2012 p. 4]. Beyond basic functions to assemble a mission plan (insert, delete, move, abstract, detail activities), timeline-based mission planning systems generally check a given plan for constraint violations. The tools differ most regarding further computational services, like "automated planning", "autonomous planning", "constraint enforcement", "external constraint models", or "soft constraints", which shall help the mission planning engineer to search for a valid mission plan (cf. [Chien et al., 2012 Tejo et al., 2006 Bresina and Morris, 2006 Aghevli et al., 2011]).

On the far other end of the spectrum of mission planning solutions, there are up to today custom tailored and mission specific tool sets of low complexity. Those are mostly used by missions which are as a whole less complex and which are not supported by the large mission operations facilities by the agencies but small entities, like universities. They range from textual procedures only, to spreadsheets and scripts to generate command sequences, and to sets of loosely coupled command line tools in combination with professional software, e.g. for orbit propagation (cf. [Lenzen, 2015 p. 168]). Those MPTs are often not even worth a publication. For small missions one often decides to omit the usage of one of the available complex systems as the effort for training and setup is much higher than the development effort of an own toolset. But this obviously contradicts the original reasons for the development of generic MPTs.

Ultimately, this unbalance in required effort for the usage of an available generic MPT and for the development of a new one is the reason for this thesis work. However, given the reasoning for generic MPTs and the great conceptual work of the fathers of the existing complex systems (mainly Smith et al., 1998 Chien et al., 2000 GSOC-DLR, 2010), combined with the strong arguments to separate the mission modelling from the core functions also for SONATE as outlined before (cf. [1]), this work aims at the development of a generic timeline-based MPT which at the same time keeps the complexity in general low to stay applicable for small missions.
At the beginning of this master’s project, I defined its scope in close cooperation with the designated users by a project statement, which is directly derived from requirements of the SONATE mission\(^1\). This project statement then leads to several primary and secondary project objectives, which I further specified by requirements. As every element of this specification is directly related to at least one of the abstraction level above, the whole specification transitively follows directly from the mission statement of SONATE.

### 3.1 Project Statement

<table>
<thead>
<tr>
<th>ID</th>
<th>Project Statement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-PS-0010</td>
<td>A mission planning tool for the SONATE mission is developed in the scope of this master’s project.</td>
<td>MB-0020 2MZ-0010</td>
</tr>
</tbody>
</table>

**Comment:**
MB-0020 requires the development of a mission planning tool for SONATE. It shall be fulfilled with this sub-work-package which fits the scope of a master’s project. Thus, also 2MZ-0010 – involvement of students in the SONATE mission is partly fulfilled with this project.

### 3.2 Project Objectives

**Note:** The order the objectives are presented here, directly resembles the importance of the features of the mission planning tool and are thus prioritized accordingly throughout the project. While the primary objectives have similar (although different) priorities, the secondary objectives are handled with much less priority.

---

\(^1\)For reference of the corresponding requirements please regard the relevant documents of SONATE.
### 3.2.1 Primary Objectives

<table>
<thead>
<tr>
<th>ID</th>
<th>Primary Project Objective</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-1PO-0020</td>
<td>Design and implementation of a procedure editor.</td>
<td>MPT-PS-0010</td>
</tr>
<tr>
<td>MPT-1PO-0030</td>
<td>Design and implementation of a scheduling timeline tool.</td>
<td>MPT-PS-0010</td>
</tr>
<tr>
<td>MPT-1PO-0040</td>
<td>Design and implementation of a command list generator.</td>
<td>MPT-PS-0010</td>
</tr>
<tr>
<td>MPT-1PO-0050</td>
<td>Design and implementation of a graphical user interface.</td>
<td>MPT-PS-0010</td>
</tr>
</tbody>
</table>

**Comment:**
To sufficiently support the mission operations, an intuitive and ergonomic machine interface is essential.

<table>
<thead>
<tr>
<th>ID</th>
<th>Primary Project Objective</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-1PO-0060</td>
<td>Development and design of concepts to face a highly autonomous satellite from the point of view of mission planning.</td>
<td>MPT-PS-0010</td>
</tr>
</tbody>
</table>

**Comment:**
SONATE is a technology demonstration mission for a highly autonomous satellite system. This might have severe consequences on the design of the mission planning tool, which might thus differ a lot from existing “standard” mission planning tools. For instance in full autonomous operation of ASAP-Light\(^2\), the satellite is commanded by ASAP-Light. A consequence is that, from the point of view of the mission planning tool in a standard design, the satellite is in an unknown state (regarding resources like attitude, battery fill-level, amount of collected data), while it is crucial for the planning of further activities to estimate the satellite’s state as exact as possible to not needlessly put additional risk on the satellite. Therefore, new concepts to face this situation have to be developed\(^3\).

\(^2\) Autonomous Sensor and Planning System - Light is one of the main payloads of SONATE, refer to [Balagurin et al., 2016](#) for further information.

\(^3\) Refer to chapter [ ] for an elaborate explanation of the problem as well as proposed concepts.
3.2.2 Secondary Objectives

It was obvious from a very early stage of the project that most of the secondary objectives are far beyond the scope of a master thesis project and that at most parts of them could possibly be reached. Nevertheless, they were properly defined and also regarded during the requirement definition, as well as the first steps of the analysis (cf. [4]) and design (cf. [6]) to keep the vision of a powerful MPT and in order to allow future extensions and keep the needed interfaces open.

<table>
<thead>
<tr>
<th>ID</th>
<th>Secondary Project Objective</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-1PO-0010</td>
<td>Design and implementation of an orbit and ground contact propagator.</td>
<td>MPT-PS-0010</td>
</tr>
</tbody>
</table>

Comment:
To plan and schedule ground contacts.
This secondary objective was originally classified as primary objective and moved to the secondary objectives only at a late stage of the project. This decision is justified by the fact, that the functionalities of this component actually are not essential for the MPT for SONATE as other tools are available to predict ground contacts. Furthermore, contact periods can be seen as resources, i.e. the design and implementation of this component has a lot in common with the resource control functions and should therefore be handled with similar priority.

<table>
<thead>
<tr>
<th>ID</th>
<th>Secondary Project Objective</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-2PO-0010</td>
<td>Design and implementation of resource control functions for the mission planning tool.</td>
<td>MPT-PS-0010</td>
</tr>
</tbody>
</table>

Comment:
To further reduce the risk on the mission, the mission planning tool shall provide functions to automatically control resources, e.g. warn the operator if a planned activity is likely to exceed the power budget. As compliance of the resources could basically also be achieved using telemetry and through planning of the operator, it is not regarded essential for the mission planning tool.

<table>
<thead>
<tr>
<th>ID</th>
<th>Secondary Project Objective</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-2PO-0020</td>
<td>Design and implementation of validation functions for user generated procedures.</td>
<td>MPT-PS-0010</td>
</tr>
</tbody>
</table>

Comment:
To automatically check the procedures for plausibility (e.g. before an instrument can be used, it has to be switched on; switch off all instruments at the end of a procedure again) and other rules. Similarly to MPT-2PO-0010, those functions would add safety to the operations, but are not considered essential.
## 3.3 Constraints

This master’s project is as every other project not only bound to its objectives but also influenced by external constraints. Those are listed here. Some constraints, mission objectives or top level requirements of SONATE act like constraints upon the development of the MPT. They are listed besides other project specific constraints and aliased for consistency.

<table>
<thead>
<tr>
<th>ID</th>
<th>Constraint</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-C-0010</td>
<td>The timeframe of the project is 6 months, beginning from 01 Feb 2017.</td>
<td></td>
</tr>
</tbody>
</table>

**Comment:**
According to the examinations regulations of the Space Master program (PO\(^5\) version 2012), the master’s project is to be conducted throughout 6 months. This project was registered officially on 01 Feb 2017.

<table>
<thead>
<tr>
<th>ID</th>
<th>Constraint</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-C-0020</td>
<td>The final software product of the MPT shall be platform independent.</td>
<td></td>
</tr>
</tbody>
</table>

**Comment:**
As various parts of the ground segment are still under development, it is not yet precisely specified which systems (operating systems, configuration, etc.) the MPT will run on. Furthermore, in the future it might be reused and deployed for other missions with even different ground segments.

---

\(^4\) This judgement origins from the beginning of the project and is quoted here as part of the project definition although the implementation effort for the developed concepts is roughly estimated (cf. \(^5\)).

\(^5\) Prüfungsordnung, German for examination regulations.
3.3. Constraints

<table>
<thead>
<tr>
<th>ID</th>
<th>Constraint</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-C-0030</td>
<td>The final software product of the MPT shall be executed on a fat client.</td>
<td></td>
</tr>
</tbody>
</table>

**Comment:**
Commonly mission control centres are equipped with recent desktop PCs. Thus, there is no need to optimize the execution of the MPT on thin clients or even less powerful systems. Vice versa, the MPT shall not require special dedicated hardware.

<table>
<thead>
<tr>
<th>ID</th>
<th>Constraint</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-C-0040</td>
<td>The MPT shall be scalable and open for extensions.</td>
<td>M-0030</td>
</tr>
</tbody>
</table>

**Comment:**
A larger mission might also introduce new requirements (e.g. more resources to control) to the MPT. Therefore it shall be scalable and open for future extensions. Secondly, probably not all features of secondary objectives can be implemented in the scope of this thesis. The MPT shall remain open for those extensions.

<table>
<thead>
<tr>
<th>ID</th>
<th>Constraint</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-C-0050</td>
<td>The MPT shall be able to plan missions with a lifecycle of at least 1 year.</td>
<td>M-0010</td>
</tr>
</tbody>
</table>

MPT-C-0060 was deleted as the corresponding requirement of SONATE was changed.

<table>
<thead>
<tr>
<th>ID</th>
<th>Constraint</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-C-0070</td>
<td>The MPT shall be executed in a mission control center at University of Würzburg.</td>
<td>M-0100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Constraint</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-C-0080</td>
<td>The MPT shall save all data in human readable data formats.</td>
<td></td>
</tr>
</tbody>
</table>

**Comment:**
It is essential for the mission, that the data, especially procedures and already scheduled activities, are not lost in case the MPT does not work anymore or is no longer supported by the systems provided in the mission control center. Therefore, it shall be saved in human readable data formats (i.e. XML, plaintext).
### 3.4 Requirements

The requirements finally specify every detail, which is important to meet the project objectives. For this master’s project they are categorized into functional, performance, design and operational requirements. Functional requirements define the top level functionalities of the mission planning tool directly derived from the objectives. They are further detailed by the performance and design requirements, which are not only affected by the functional requirements but also by operational requirements and external constraints (cf. 3.3).

Performance requirements specify "how good" a functionality has to be and design requirements settle some details of the design one must not decide about as they follow directly from the chain of requirements. Operational requirements are related to the actual usage of the final software product.

Requirements are phrased as full sentences. The level of obligation a requirement has to be met to ensure a functional software product fulfilling its objectives, is classified by several keywords:\(^6\):

**shall** The requirement is essential and must be implemented.

**should** The requirement is a recommendation. Without its implementation the system will become less effective/efficient.

---

\(^6\)The definitions of the keywords are adapted from the SONATE project.
3.4. Requirements

**may** The requirement grants permission to the system, possibly with a binding limitation.

**can** The requirement is a possibility or capability of the system which makes it more attractive.

To non-essential requirements (should/can), a priority can be assigned to indicate the importance of the requirement. There are the levels **high**, **medium**, and **low**.

The complete set of requirements is given in appendix A.
This page is intentionally left blank.
The next step of a software project after the project definition is commonly the analysis, i.e. the clarification of the implications of the requirements on the future design of the system. In this project, an object-oriented approach using UML-notation\textsuperscript{1} is applied and the methodology of [Oestereich, 2012] is adapted in large parts.

In a first step system use cases (cf. 4.1), i.e. "what different users would like to do with the system", are identified and described. Afterwards the relevant subject-specific entities and their relations are named, described and summarized in a subject-specific class diagram (cf. 4.2). As the last step of the analysis, the interfaces towards external systems and the users are defined (cf. 4.3).

Throughout the object-oriented analysis (and partly also during the design, cf. 5) special attention was directed to a system, which fulfills not only the primary objectives but also each and every of the secondary ones. This eventually leads to the fact, that especially in the scope of this chapter use cases and features are discussed, which were not further pursued in the scope of this thesis due to their low priority and are thus dropped in the discussion of later chapters. To distinguish those parts from the ones which are connected to the primary objectives and are essential for the success of the thesis project, the essential parts are marked green in relevant figures.

Please note as further remark, that the complete analysis is a highly iterative process. The presented order is the one conducted in the first iteration and providing the best overview of the system from very abstract to more and more detailed descriptions. Nevertheless, due to the further iterations, the following sections influence each other not stringently and linearly, causing several cross-references within this chapter.

\textsuperscript{1}Unified Modeling Language
4.1 System Use Cases

4.1.1 System Context

To facilitate the identification of system use cases, the system context was acquired. It shows which users and which external systems interact with the system and on a very abstract level also which data is exchanged in which direction. Figure 4.1 shows the system context of the MPT for SONATE. A Systems Engineer, i.e. an expert in the field of the satellite, its capabilities and limitations, as well as the complete mission environment, provides the mission model. Some Flight Dynamics System provides orbit parameters, most probable in the form of two-line-elements (TLE). The Telecommand Database provides the available telecommands (TC) including parameter number, types, and limits. A Principal Investigator being in charge for a payload on SONATE puts experiment requests to the Planning Engineer, while the Flight Director being in charge for the satellite (bus) in general puts maintenance requests to him. The systems engineer might request to conduct some further activities, i.e. to do system checkouts or verify the system model. The planning engineer combines those three types of requests and raises in more abstract terms planning requests to the MPT. As a product the MPT provides a human readable sequence of events (SoE) to the Operator which helps him to monitor the operations and check if everything goes according to the mission plan. Furthermore, the MPT provides the SoE to the Telecommand Client. The SoE provided to the telecommand client is basically a subset of the SoE provided to the operator as it only contains immediate or time-tagged TCs, but no other information like steps to be performed on ground or explanatory remarks, and is encoded in a machine format readable for the telecommand client.

![Figure 4.1: System context diagram.](image-url)
This general context at hand, the system use cases can be identified quite easily. They are shown in figures 4.2, 4.3, and 4.4 and explained in the following sections.

4.1.2 Use Cases of the Systems Engineer

As the systems engineer (cf. figure 4.2) is supposed to provide the mission model, his overall use of the MPT is to model the mission. This abstract task can be separated in three main use cases. The systems engineer wants to use the MPT to model a resource, to model a procedure, and to model a constraint\(^2\). Each of those use cases can further be detailed\(^3\) as, in order to model a resource, procedure, or constraint respectively, the systems engineer wants to either create a new one, or alter an existing one, or delete an existing one. As a constraint is either defined on the scope of a TC, a procedure or globally on a resource, different use cases to model a constraint exist for each of those possibilities. Last but not least, the MPT shall support the systems engineer to ensure that the procedures he has modelled are semantically correct. Hence, he wants to validate a procedure by the means of the MPT.

4.1.3 Use Cases of the Planning Engineer

While the systems engineer provides the mission model, the planning engineer (cf. figure 4.3) uses it to plan the activities of the mission. He therefore wants to update a resource to feedback the mission model of the MPT with valid and up-to-date data

\(^2\)For proper definitions of the terms "resource", "procedure" and "constraint" refer to 4.2

\(^3\)This is indicated in the diagrams according to the UML-notation by the dashed arrows pointing to the more general use case.
about the state of the mission. Given the current and past values of a resource, he wants to propagate a resource to get information about its future course. The planning engineer’s main use of the MPT is of course to schedule a timeline entry\(^4\), which can either be to insert a new timeline entry, to move an existing one, or to delete an existing one. While scheduling probably several activities, the planning engineer regularly wants to inspect the mission plan, to retrieve information about the scheduled activities and their effects on resources. Similarly to the validation of a procedure the systems engineer is conducting, also the planning engineer wants to validate the mission plan before it is approved for execution. To provide the operator and the telecommand client with the data which is necessary for the execution of the mission plan, the planning engineer wants to generate the sequence of events. That includes the more detailed use cases of the generation of the sequence of time-tagged telecommands, a sequence of immediate telecommands (both in a machine format of the telecommand client), the full set of information of the SoE, and the ungrounded sequence of events of a procedure\(^5\). With increasing duration of the mission also the complete mission plan grows in size and complexity. To prevent negative effects on the performance of the MPT regarding the handling of the mission plan, from time to time the planning engineer wants to clean the mission plan, i.e. to archive past activities and the past course of resources over time, which are no longer relevant for future planning.

Figure 4.3: Use case diagram for the planning engineer.

---

\(^4\)For the definition of "timeline entry" refer to 4.2.

\(^5\)This is the template of a procedure not fixed to a specific point in time, but only relative to the start of the procedure, which can for instance be used to conduct the procedure in an immediate instead of time-tagged manner.
4.1.4 Use Cases of the Operator

Last but not least, the operator’s sole use of the MPT is to monitor the currently executed sequence of events in an interactive way (cf. figure 4.4).

![Operator diagram](image)

Figure 4.4: Use case diagram for the operator.

4.1.5 General Remarks on the Use Cases

Although all those use cases basically can be executed independently from each other and in an arbitrary multiplicity and order, there are several logical dependencies between them. For instance, before the planning engineer can schedule a timeline entry, the corresponding procedure has to be at least created and should preferably be fully modelled by the systems engineer. Thus, it is recommended to regard the following rules in order to prevent unnecessary inconvenience or even mistakes and errors in the planning process.

1. The complete modelling of the mission (i.e. the systems engineer’s work) should be done and frozen, before the planning engineer starts his work.

2. The telecommand database should be frozen before telecommand-specific constraints and procedures are modelled.

3. The systems engineer should first model all resources together with the corresponding global resource-specific constraints, then the telecommand-specific constraints, and finally procedures together with the corresponding procedure-specific constraints.

4. Before propagating a resource the planning engineer should ensure that the resource is sufficiently up-to-date\(^6\).

5. The planning engineer should inspect the mission plan, schedule a timeline entry, propagate all resources and validate the mission plan in that order and in an iterative manner.

\(^6\)That means that he does not necessarily have to update it every time before a propagation is run, but depending on the quality of the propagation e.g. on a daily or bi-daily basis.
6. Before generating the sequence of events the planning engineer shall validate the mission plan.

7. The operator shall ensure that the sequence of events he received from the MPT is up to date and matches the most recent version of the mission plan.

### 4.2 Subject Specific Classes

Based on the work of [GSOC-DLR, 2010] and [Smith et al., 1998], I model the general problem of mission planning with the following subject specific classes. They are shown in figure 4.5 and explained respectively defined in the following paragraphs.

The main hierarchy is quite simple and straight-forward. The atomic unit of the mission plan is a *Step*. The *Step* contains information, like commands to the operator, values to be read from telemetry, as well as optionally one *Telecommand* to be executed on the space segment. A *Procedure* consists of a well-ordered and well-timed set of *Steps* and accompanying meta-data (like description, preconditions, post-conditions, pass/fail criteria, duration etc.). A *TimelineEntry* is a specific manifestation\(^7\) of a *Procedure* which is scheduled to be executed at a grounded absolute time point. Regarding the association between *TimelineEntry* and *Procedure* the other way round, one could also say, that the *Procedure* is the "template" or "stencil" for an *TimelineEntry*. Finally, the complete *MissionPlan*, which is a singleton object, contains all scheduled instances of *TimelineEntry*. This hierarchy is sufficient for the modelling of *Procedures* and the actual, however completely manual and unassisted, mission planning as required in the primary objectives (cf. 3).

All other classes are related to the features of orbit and ground contact propagation, resource control, and procedure validation, which are classified as secondary objectives only. A *Resource* is an abstract class which represents some (physical) state of the mission. *NumericResource* and *FiniteStateResource* are the elementary extensions of the abstract *Resource* class. In most cases, a resource will probably represent a state of the space segment, i.e. the fill-level of batteries (*NumericResource*) or the power-status of an instrument (*FiniteStateResource*), but is not limited to it and could basically also represent the ground segment, e.g. the availability of ground stations or operational staff. A *BooleanResource* is modelled as a special case of a *FiniteStateResource*. In general, simple resources can directly be modelled by the systems engineer (cf. 4.1), but more complex ones can separately be implemented by sub-classing the corresponding *Resource* class and, where applicable, provide their own means for modelling and configuration by the systems engineer. This is the case for the resources *GroundContact* and *Eclipse* which propagate the availability of ground contact to a specific *GroundStation* for a satellite in a specific *Orbit*, respectively the eclipse- and day-times of the satellite. As the MPT shall primarily support SONATE which is a single-satellite mission, the *Orbit* is modelled as singleton.

---

\(^7\)I avoid the term "instance" as it could be misunderstood in the context of object oriented development.
In fact, the model becomes even more capable, if resources could also be modelled by the systems engineer as CompositeResource following the composite design pattern, e.g. to model the numeric resource of the total available power $P_{tot}$ as $P_{tot} = P_{bat} + ((\neg(Eclipse)) \land (Attitude == SUNPOINTING)) \times P_{sol}$, where $P_{bat}$ and $P_{sol}$ are NumericResources representing the power available from the batteries and solar panels respectively, and $Eclipse$ is a BooleanResource and $Attitude$ a FiniteStateResource. However, this feature exceeds even the scope of the secondary objectives is hence left out completely for reasons of simplicity\(^8\) and where needed composite resources could be implemented directly as extensions of the basic resources.

A Resource can be linked to a Procedure, to a Step, or to a Telecommand through a ResourceConstraint\(^9\). Staying quite close to the model of [GSOC-DLR, 2010], a ResourceConstraint can either be a ResourceComparison, a ResourceBound, or a ResourceModification, where the latter is further detailed into SetModification and DeltaModification. A DeltaModification is again sub-classed by the more detailed types ResourceAllocation and ResourceAccumulation. Note that, although only seldom semantically meaningful for NumericResources (e.g. to "set" the fill-level of all data-storages to zero when a "clear all storages" command is executed), a SetModification is allowed for any type of Resource, while a DeltaModification is restricted to NumericResources as the underlying operations of addition and subtraction are not defined on finite states. All types of ResourceConstraint could also globally act on a resource without a link to a specific Procedure, Step or Telecommand.

Constraints which link a Procedure to itself, another or several other Procedures are DependencyConstraints. Such a constraint could either be a TimeDependency, a Demand, or an AssignmentRestriction. The same types of constraints also apply to link a Telecommand to itself, another or several other Telecommands.

As all constraints shall finally be equally handled by the control logic, all types implement the interface Constraint. Please note, that it would render the model more powerful, if all types of constraints were allowed on all levels of abstraction (i.e. on TimelineEntry, Procedure, Step and Telecommand). Like that an eventually scheduled TimelineEntry would have its own "local" Constraints, inherit the ones from the underlying Procedure, which itself brings besides own ones the ones inherited from the Steps, which in turn could inherit ones from the associated Telecommand. However, this is omitted for reasons of simplicity in this model, as the semantic meaning and use cases for the excluded combinations of types of constraints and abstraction levels are considered to be very limited and unlikely. The proper definition of all types of constraints is not elaborated, as the complete resource control remained unimplemented in the scope of this thesis. Please refer to [GSOC-DLR, 2010] and [Lenzen, 2015] for more detailed explanations of the different types of constraints.

\(^8\)This is indicated by crossing out the class name in figure 4.5.

\(^9\)Note that the corresponding associations are left out in figure 4.5 for simplicity.
Figure 4.5: Class diagram for subject specific classes.
4.3 Definition of System Interfaces

4.3.1 Data Interfaces

Telecommand Database

The set of available telecommands and their parameters is defined in a SQL-database\(^{10}\). The MPT shall use this database to suggest valid telecommands to the systems engineer while modelling procedures and to validate procedures. Thus, the telecommand database, whose total size ranges in the order of < 1 MB, will be asynchronously accessed as input interface and will probably be loaded once at the start-up of the MPT.

Flight Dynamics

The flight dynamics system, which is from the MPT’s point of view a black box, provides current orbit parameters. It can either provide them in a human readable format or as TLE. If they are given in a human readable format, the MPT shall provide an user interface to enable direct input by the systems engineer. If the orbit parameters are to be given as TLE, the MPT shall automatically collect them either from a local file or an online resource, e.g. CelesTrak (cf. [Kelso, 2017]). Also the interface to the flight dynamics system is an asynchronous input interface and will be accessed at the startup of the MPT and about once a day if the MPT runs longer than a day, resulting in a very low data volume (< 1 kB/day).

Telecommand Client

The MPT’s only direct data output is the sequence of telecommands, either time-tagged or immediate ones. The MPT provides them asynchronously as plain-text files (size in the order of some kB) when the generation is queried by the planning engineer (about once a day). The format is defined by the telecommand client (cf. [Beylin, 2014]).

4.3.2 Graphical User Interface

As the use cases of the systems engineer, the planning engineer, and the operator are quite different (cf. 4.1), also their requirements and needs for the graphical user interface (GUI) differ a lot. Therefore, it is reasonable to also split the definition of the GUI in those three major parts, the Planning Perspective, the Modelling Perspective, and the Monitoring Perspective. The contents and basic structure of the key elements of those perspectives were developed as pencil sketches as a starting point for the further design (cf. figures 4.6, 4.7, 4.8). The general appearance is of course influenced a lot by successful state of the art timeline-based MPTs (cf. 2). Nevertheless, the key elements shall be briefly discussed here.

\(^{10}\)The format of the database is defined in [Beylin, 2014].
Another key factor for the design of the GUI appeared already during analysis and might be critical for the overall success of the mission. Especially for non-nominal cases it might be essential for the planning engineer to keep full control over the planning process and possibly even override existing rules which are reasonable and helpful during nominal mission phases but do not apply in non-nominal cases. Thus, the MPT and especially the GUI must always provide full control to the user even if assisting algorithms like the resource control functions or logical checks for user input conclude that the user action seems not to be reasonable. A simple example would be the scheduling of a timeline entry whose start date is already in the past. In most cases this is not reasonable or even might be a threat to the mission as it might mess up the mission plan. The assisting algorithms may justly assume that changing a timeline entry in the past is a user mistake and warn the user about that. However, the user interface must not prevent the conduction of this user-invoked action. For instance it might be the off-nominal case that the satellite’s clock lost synchronization to the clocks on ground and it might be essential to conduct a procedure which is already in the past compared to the clocks on ground to face this off-nominal situation. The need to fiddle around and cheat the own MPT in such a, possibly even time critical, situation poses a major risk on the overall mission. Thus, the user interface must always provide intuitive ways to opt against the advice of the assisting algorithms.

Planning Perspective

The heart of the MPT is the planning perspective (cf. figure 4.6). It empowers the planning engineer, who will be the main user, to conduct all his use cases. While the upper part shows some general information, like the current time, and provides controls, the actual timelines take the better part of the screen. The planning engineer can simultaneously inspect the plan by scrolling along the time axis at different zoom levels and insert, delete or change timeline entries on the timelines. The appearance of the timelines, e.g. which clocks, resources, subsystems, or procedures to be shown, can be configured on the left hand side. Details of any timeline might be shown as tooltip when hovering the cursor over relevant objects, as well as in a dedicated display in the lower part of the window. Any problems might be indicated by symbols attached to the corresponding object and in a problem console.

Besides the planning engineer, also the operator can use this perspective to monitor the current activities. To do so, he shall lock the controls to prevent unintended changes of the mission plan. The current time is indicated for the operator in the timelines as vertical line and the operator might put the timelines to auto-scroll thus that the current time remains at a fixed position on the screen while the timelines traverse from right to left as time passes.
4.3. Definition of System Interfaces

Figure 4.6: Pencil sketch of the GUI prototype of the planning perspective.

Modelling Perspective

The main element of the modelling perspective is the procedure editor (cf. figure 4.7). The systems engineer can create and edit procedures here. The editor provides areas to edit general information about the procedure, like the description, preconditions or pass/fail criteria and another area to define constraints. The table for the sequence of events takes the better part of the editor window. Here steps can be added or deleted, as well as existing ones changed. Similarly to the planning perspective, problems are indicated with symbols and in a problem console. Several actions, like the definition of constraints, might be conducted through additional dialog windows and the GUI might provide features like auto-completion for telecommands to facilitate the work of the systems engineer. The modelling perspective can hold several instances of the procedure editor and provides assisting GUI-elements, like a explorer view, to open and close the former.
Monitoring Perspective

Compared to the other perspectives the monitoring perspective (cf. figure 4.8) is rather simple as it has to realise only one use case. It consists of a top area showing the current time, and three tables take the rest of the window. The tables show the past, current and upcoming steps from all scheduled timeline entries in the correct order. Besides the information of a step defined in the procedure, the name of the procedure where the step originates from as well as timestamps and countdowns for its start and end are included in the tables. They are automatically updated, i.e. a step moved from the table of upcoming steps to the one for current ones and finally for past ones as time passes.

To monitor the mission plan, the operator may also refer to the planning perspective (cf. 4.3.2).
4.3.3 Other User Interfaces

The MPT shall provide the sequence of events not only in the machine readable format and on screen, but also as human readable file. The sequence of events, containing all data of the most recent version of the mission plan, is hence saved as comma-separated text file (which can easily be imported to other tools, e.g. Microsoft Excel for further formatting and processing) when queried by the planning engineer. It is expected that the file size range in the order of tens of kilobytes and that the files are created about once a day.
This page is intentionally left blank.
5.1 General Problem

Although the field of highly autonomous systems in the space sector is growing fast in the recent years (cf. [Wojtkowiak et al., 2013, Wörle and Lenzen, 2013, Chien et al., 2013]), only little to no publications about the implications such systems pose to the mission operations and especially the mission planning process can be found. Only [Wille et al., 2013] acknowledges that there are implications which significantly influence the planning process but does not detail how to face those new circumstances from the point of view of mission planning.

Visionaries might argue that on long term the classical elements of mission operations and mission planning will vanish as the main tasks of coordination and control will be conducted autonomously by the satellites. However, keeping the conservative general attitude, which governs the space sector to keep risks as low as possible, in mind and regarding the complexity of resources like ground stations which are shared between missions and would require an autonomous cross-mission coordination, one has to admit that this vision will not come true without a significant transition period. Especially at the current state of research, where highly autonomous satellites are flown as technology demonstration missions or highly autonomous payloads are embedded together with non-autonomous payloads into one mission, basically the "best of both worlds" - from classical mission operations and mission planning, as well as autonomous concepts - are required. But those two paradigms contradict each other quite a lot from the point of view of mission planning.

The classical mission planning approach, even when combined with advanced algorithms for resource control or automatic scheduling, can basically be seen as an open-loop control: Starting from a well defined state of the satellite, the planning en-
engineer commonly schedules a set of well-timed activities, whose effects on all systems are well-known thanks to profound mission modelling, hence traversing the system to another well-defined state. The state of the art is, that even during contingencies the systems switch to a quite well-defined safe-mode and the engineers try to recover the mission to a well-defined nominal state by conducting well-defined procedures which assume the safe-mode as starting point. Depending on the models’ quality, from time to time it might be necessary to compare the predicted state with the actual one which is acquired through telemetry or external observatory systems in order to correct accumulated deviations of the model. This is however not regarded as part of the daily standard planning process.

The picture is significantly different when taking also highly autonomous systems into account: When the highly autonomous systems take control over the satellite, the effects on various resources like power-budget, data storages, or even the satellite attitude are so extensive and only depend on the results of the autonomous algorithms and external events taken into account by them, that the assumption of the ground segment constantly knowing about the well-defined state of the satellite\(^1\) no longer holds true. Assuming that the autonomous systems work correctly and safely, this lack of knowledge on ground is no major problem as long as the autonomous systems remain in charge over the satellite. But it becomes crucial when switching back to phases of classical mission planning, when the planning engineer is supposed to prepare a plan for phases following an autonomous phase while not knowing the exact state of the satellite. Each and every procedure to be executed might exceed some resource and thus pose a major risk on the mission.

Because alternating autonomous and classical mission phases will be the case for most autonomous missions in the near future as outlined above, and especially the mission scenarios of SONATE (cf. [Balagurin et al., 2016]) do explicitly imply them, concepts to overcome the lack of knowledge of the planning engineer and the accompanying risks for the mission which can be included in the standard planning process are to be developed. In the following sections of this chapter I propose three of them.

The presentation of those concepts, which vary in their powerfulness and implementation efforts, includes not only an explanation but I also discuss their advantages and disadvantages.

### 5.2 Safe Planning

The safest option from the point of view of mission planning, as well as the easiest way regarding implementation efforts for the MPT to face the described problem of switching from an autonomous state back to classical operations, is to simply tell the planning engineer and discourage any activities until the actual state of the satellite was acquired. This would for instance mean, that the MPT would rise a conflict for every timeline entry which is scheduled after one which is marked

\(^1\)or at least sufficiently accurate predictions
as autonomous but before the first uplink opportunity after the next housekeeping downlink (cf. figure 5.1). The MPT could additionally suggest to automatically delete all future timeline entries when an autonomous one is inserted into the mission plan. This concept which I call *Safe Planning* guarantees that

1. no timeline entry which might exceed some resource is scheduled immediately after the autonomous phase.

2. the actual state of the satellite is acquired which is the reliable basis for the further classical planning process.

3. enough time is left to decide whether the classically planned timeline entries after an autonomous phase can indeed be conducted or have to be cancelled as the state of the satellite was different than expected.

Besides the almost neglectable implementation effort for the MPT — to mark autonomous timeline entries and trigger the corresponding conflicts and warnings — the autonomous systems would have to guarantee to spare enough resources for the first housekeeping downlink after the autonomous activity.

While this concept provides a high level of safety for the overall mission and is easy to implement, it obviously degrades the overall mission efficiency and utilisation significantly. After each autonomous phase, as short as it might be, the satellite idles until at least two ground contacts (one for down-, another for uplink) were established (cf. figure 5.1). Depending on the orbit and availability of ground stations this idle time can range in the order of a day\(^2\). This might be acceptable for technology demonstration missions whose primary objective is to demonstrate the reliability of the autonomous components. But it is definitely not acceptable for missions with multiple scientific payloads which would have to idle as well. And it would absolutely contradict the original motivation for highly autonomous satellites, which is to increase efficiency and scientific output (cf. for example [Chien et al., 2005]), if the autonomous systems were to be utilized during routine operations.

![Figure 5.1: Example for Safe Planning. The red classically planned timeline entry causes a conflict as it is in the idle time. The first uplink opportunity cannot be used as it is not after the first downlink.](image)

\(^2\)assuming LEO and only one ground station, as it would be the case for SONATE
5.3 Sigma Resource Propagation

If there are resource control functions available, there is another concept to face the problem of alternating phases of autonomous and classical mission planning which I call *Sigma Resource Propagation*. This concept links the idea of statistic error propagation to the problem to predict the satellite’s state during phases of autonomous operations. When in previous sections it was claimed that autonomous operations leave the satellite in an "undefined" state, that is not completely true. Of course, also every autonomous system has upper and lower boundaries regarding resource consumption and similarly the transitions for finite state resources are limited. Thus, speaking in terms of statistic error propagation, after a phase of autonomous operations the satellite actually does not end up in an undefined state, but in a defined state somewhere in the proximity of the "most probable state". Due to model imperfections this is actually also the case for classical operations. But in contrast to them, the "standard deviation" of the satellite’s actual state around the most probable state is much larger for autonomous activities, which ultimately renders subsequent classical planning impossible or risky.

To illustrate this perspective on the problem, imagine a satellite with an arbitrary scientific instrument which can take data once a minute and each data-take consumes commonly $1.5 \text{ MB}$ of the non-undefined data-storage. For some arbitrary reasons, it might sometimes consume only $1.4 \text{ MB}$, sometimes $1.6 \text{ MB}^3$. If this instrument is commanded in a classical way to take samples for one hour with a sample rate of one sample per minute, this results in the most probable value of consumed data-storage of $60 \times 1 \text{ sample/min} \times 1.5 \text{ MB/sample} = 90 \text{ MB}$. However, the actual fill level of the data-storage may vary by $\pm 6 \text{ MB}$ due to the model imperfection. This does not obstruct further classical planning, as the error is one order of magnitude smaller than the total amount of data and in the order of only few data-takes$^4$. Now imagine the very same instrument not being controlled classically but by an autonomous system for the very same period of one hour. But in contrast to the classically conducted procedure the autonomous system is allowed to dynamically decide for each data-take, depending on external events, whether it shall be stored or discarded. After said hour of autonomous operations hence the fill level of the data-storage is anywhere between 0 MB and 96 MB with its most probable value (depending on the statistics of the external events and the exact rules of the autonomous system) perhaps right in the middle at 48 MB. As the difference between the lower and upper bound is in the order of several magnitudes, it has to be regarded for subsequent classical planning runs. Whether this difference poses any threat of overexploitation of the resource, however depends on the overall fill level of the data-storage and the consumption of further activities to be classically scheduled.

The main idea of the Sigma Resource Propagation is to model all three, the most probable value as well as the upper and lower limits, as effect of a timeline entry

---

$^3$This variance is heavily exaggerated for the sake of the example.

$^4$With less exaggerated deviation it might even be completely neglectable.
on the corresponding resource. Additionally, the resource control algorithms would propagate the most probable state of a resource independently from the upper and lower limit of possible values (cf. figure 5.2).

When a new timeline entry shall classically be inserted into the mission plan, the resource control checks the lowest possible, most probable, and highest possible propagated state of the resource against the lowest possible, most probable, and highest possible effect of the timeline entry. Depending on the results of those $3 \times 3$ tests, the resource control acknowledges the insertion of the new timeline entry or raises conflicts of different severity. To illustrate this behaviour, we return to the example discussed above. Now we additionally define the overall size of the data-storage to be 135 MB and that it was completely empty at the beginning of the example. Furthermore the most probable value of the autonomous operations of one hour shall be at 48 MB.

In case 1 (cf. figure 5.3) a timeline entry of one hour classical sampling was already scheduled. In case 2 (cf. figure 5.4) a timeline entry of one hour autonomous sampling was already scheduled.

If the planning engineer now tries to insert another hour of classical sampling in case 1, the resource control raises a conflict of highest severity as all combinations of possible states exceed the limit. If another hour of classical sampling shall be inserted in case 2, the resource control raises again a conflict but of lower severity as in case 1 because there are combinations of possible states that do not exceed the limit.

If the planning engineer tries to insert half an hour of classical sampling, again conflicts will be raised in both cases. But again the conflict of case 2 is of lower severity as for case 1 because it is less probable that the limit will be exceeded.

If the planning engineer tries to insert 15 min of classical sampling, the resource control will accept the request, as even in the worst case the limit will not be exceeded.

This approach to track all possible values, which are the values between the lower and upper bound, is quite straightforward as seen from the example for numeric resources. It may even be implemented by means of the mission modelling only and

![Sigma Resource Propagation](image)

Figure 5.2: Example for the sigma propagation of an arbitrary resource.
Figure 5.3: Example for Sigma Resource Propagation. Case 1. First data-take: green – lowest possible; black – most probable; magenta – highest possible. Projection of second data-take: colors according to the start of the projection. Line styles: dashed – lowest possible; solid – most probable; dotted-dashed – highest possible.

without any further implementation effort for the core functionalities of the MPT. The systems engineer would only have to model for each actual numeric resource \( x \) the three resources \( x_{\text{min}}, x_{\text{avg}}, \) and \( x_{\text{max}} \) as well as for every procedure the corresponding \( _{\text{min}}, _{\text{avg}}, \) and \( _{\text{max}} \) effects on those resources. Although this is basically possible, the modelling task would be much more convenient if the MPT natively supported the Sigma Resource Propagation. But this would of course add implementation effort during the development of the MPT.

While the concept itself can easily be understood as well as the implementation is straightforward for numeric resources, this is not the case for finite state resources. This is due to the fact that the set of possible values cannot be achieved by simple arithmetics. However, a concept, similar to the one of nondeterministic finite state machines known from theoretical computation theory (cf. [Sipser, 2006, ch. 1.2]), is the keyword to overcome this problem. When propagating the finite state resource, the resource control starting from a deterministic state has to consider all possible state transitions and assign a probability to each one. In the next step, all states with a probability different from 0 and their weighted possible transitions have to be
considered, thus building a tree of all possible states. If two transitions from different states lead to the same state their probabilities simply have to be added. The tests of the resource control to determine if there is a conflict then have to check if the desired state is element of the set of possible states or not. To further detail the algorithm’s conclusion it shall also take the cardinality of the set of possible states and the probabilities of the states to be reached into account.

Figure 5.6 illustrates such a propagating calculation for the state machine of figure 5.5 which could for instance represent the states of a camera-cover opening mechanism which takes two discrete time steps to open and close the cover. The autonomous system might at any time decide (with a chance of 50%) whether to open or close the cover and could also abort (with a probability of 20%) the ongoing opening respectively closing process.

To conclude this section about the concept of Sigma Resource Propagation the main advantages and disadvantages shall be summarized briefly. As it can be seen from the design of the concept, it provides a quite powerful method to check whether a classically scheduled timeline entry poses the risk of overexploitation of resources on the mission when taking place after an autonomous phase. Thus, the aim to assemble valid and safe mission plans while maximizing the overall utilisation can
Figure 5.5: Example of finite state machine of a camera-cover opening mechanism. Edges represent possible state transitions, weighted by their probability for autonomous operations.

Figure 5.6: Propagation calculation for the finite state machine (cf. figure 5.5) for three discrete time steps. The propagation of the autonomous phase starts from the well-defined state "closed". The edges are weighted as in figure 5.5; the states assigned their total probability, the most probable state is highlighted in green.
be achieved despite of the problems arising with alternating phases of classical and autonomous operations. Furthermore, the concept could basically also be applied to classically scheduled activities to reduce the risk of imperfect modelling. This might be especially beneficial for resources which are hard to model like the thermal budget. Nevertheless, the concept remains an open-loop approach for the planning process and is thus highly depending on the quality of the model. If no feedback of the actual state could be given at all (i.e. also not in sporadic intervals, as they would not be regarded as part of the standard planning process), the possible (upper and lower) bounds of the resources would diverge with time to their absolute maxima and minima respectively and thus would contain no data which can reasonably be used to improve the mission planning regarding risks and efficiency. The main disadvantage is the significantly increased effort for the development (or at least modelling) and it has to be thoroughly investigated for each mission if the benefits of a higher utilisation of the satellite justify it.

5.4 Direct Telemetry Feedback

Obviously, the solution to overcome the problems implied by the lack of information of the planning engineer, which is closest to the root cause, is to simply provide this information to the planning engineer in the standard planning process. The Direct Telemetry Feedback into the MPT and thus onto the desk of the planning engineer would, so to say, transform the previously described open-loop control of the classical planning approach into a closed-loop control. It is of course favourable, as regularly providing the most recent information to the planning engineer, significantly increases the mission safety and efficiency. This remarkably applies, similarly to the Sigma Resource Propagation (cf. 5.3), for phases of both classical and autonomous planning, while its benefits scale higher for autonomous phases as for classical planning it would "only" overcome model imperfections and no absolute lack of information.

This approach however comes along with several challenges as well. First of all there is the massive additional implementation effort for the MPT to provide and implement the data and user interfaces to the telemetry database. This especially might affect the cross-mission applicability of the MPT as the formats of the telemetry database often differ significantly from mission to mission. Secondly, one has to wisely balance which telemetry, its representation, and the period of the feedback, shall be provided to the planning engineer. The MPT must not mutate to a "telemetry monitor with planning components" and overload the planning engineer with data which is only of little relevance for him, like historic data. Similarly, too short periods for the feedback would imply that the MPT very often has to update the propagations of resources as well as validate the mission plan against the changed course of time of the resources. To perform those computational intensive tasks very often, might not only affect the MPT's overall performance, but might also obstruct the planning engineer's general workflow. If the plan is validated very often, this probably comes along with conflicts of actually minor importance being raised very often.
No vel Concepts for MPTs for Autonomous Satellites (after each re-propagation and re-validation). The planning engineer shall in general address and solve any conflicts before proceeding with further planning. Hence, the numerous telemetry feedbacks would probably require numerous minor re-scheduling actions, i.e. shifting all timeline entries by a second. This constant tinkering with the existing parts of the mission plan draws a significant amount of the planning engineer’s attention from his task to constantly push forward the planning horizon. Thirdly, while the Direct Telemetry Feedback would address the problems implied by the lack of information straight-forward for resources which are directly represented by telemetry fields, i.e. the battery status, this is not necessarily the case for all resources relevant for the mission planning. For instance, it might be relevant for the mission planning, where some instrument is pointing to in relative terms (e.g. "nadir", "horizon", "sun", "none"). If this information is not directly transmitted as telemetry, it has to be modelled as a function of the current position in orbit and the attitude angles. This does not only increase the necessary modelling effort but also the one for design and implementation to provide this functionality.

To summarize, the Direct Telemetry Feedback is regarded the most powerful concept to face the problem of alternating phases of autonomous and classical planning regarding safety and efficiency, but comes along both with massive additional implementation effort and conceptual problems which would have to thoroughly be regarded and taken into account if one pursues the concept.

5.5 Concluding Remarks

I developed three different concepts to address the problems which are implied by alternating phases of autonomous and classical planning, as they were outlined at first (cf. §5.1). I presented them here separately, but of course they could also be combined and further varied in their extend. Please note, that both Sigma Resource Propagation, and Direct Telemetry Feedback are only applicable for MPTs which feature resource control functions, while Safe Planning remains the only concept to face the problem where this is not the case. Therefore, and as the disadvantage of decreased overall efficiency is acceptable for technology demonstration missions, it is the most likely one to be implemented for SONATE. However, all concepts can be regarded as valuable starting points for further research to face the challenges posed to mission planning implied by the emergence of highly autonomous satellites.
CHAPTER 6
Object Oriented Design

After the object oriented analysis conducted in chapter 4 and the theoretical considerations about the mission planning for highly autonomous satellites (cf. 5) the design of the MPT shall be elaborated in this chapter. Firstly, the architectural pattern, which is applied, the model view controller (MVC) paradigm, is briefly explained in general in 6.1. Afterwards (cf. 6.2) the system architecture of the MPT is outlined and explained. The third section (cf. 6.3) covers the detailed design of the subject specific classes, which were already introduced in 4.2. A brief introduction to the Eclipse Rich Client Platform, which the MPT is based on, is given in the last section, as well as the arguments for this extensive design decision are outlined and the implications for the design are explained.

6.1 Model View Controller Paradigm

The model view controller (MVC) paradigm is a standard architectural pattern for the development of software applications which feature both a complex business logic (also called model) and complex (graphical) user interfaces (or called views). The main objective of the paradigm is to abstract the business logic from the user interface, thus that they can be developed independently. Hence, for instance, several user interfaces (for different groups of users and use cases, or platforms) may be developed and exchanged while the business logic stays the same. Changes and extensions of the model similarly do not necessarily require changes of the user interfaces. While the model only represents the business logic internally and the views only provide a representation of it for the user, the controller acts as agent between the model and views to translate user interactions and input to the view into changes of the model and vice versa updates the view when the model changes. Standard literature proposes several variants of this general concept, emphasizing the one or the other component, or adding further ones, e.g. for persisting the current model state (cf. [Ludewig and Lichter, 2013] pp. 433-435]).

As the MPT will be such a software application with both a complex business
logic and a complex GUI, this paradigm in general is the most promising one. The complexity of the persistence of the model, numerous procedures to be stored in XML-format, the overall mission plan to be stored as XML-file as well, several preferences and interfaces to the telecommand database and telemetry definition database, lead to the decision to separate it into another component, hence the MVC paradigm with additional persistence layer will be implemented (cf. 6.2). The decision to follow the MVC paradigm as is quasi state of the art is additionally strongly encouraged by several features of several platforms and libraries like GUI widgets and databindings (cf. 7.2.1).

### 6.2 System Architecture

Based on the object oriented analysis (cf. 4) and the MVC paradigm with additional persistence layer (cf. 6.1), I designed the general system architecture as shown in figure 6.1. Please note that this component diagram still includes components to implement all primary and secondary objectives (cf. 3) as well as the concepts for MPTs for autonomous satellites (cf. 5) although several of them will remain unimplemented due to their low priority (cf. 3).

![Figure 6.1: Diagram of the system architecture.](image)

Figure 6.1 shows the four layers Persistence Layer, Model, Control, and View. Each block in the diagram represents a component and the "adapters" indicate the pro-
vided and requested interfaces towards other components. The "male" adapters (full circle) represent interfaces which are provided by the corresponding components, the "female" ones represent requested interfaces. Where the relationship is unambiguous, the adapters may be plugged-in directly or connected with arrows and a specific naming is omitted. Where this is not the case, direct connections would obstruct the diagram, or an interface is requested by several components the corresponding adapters are distinctively labelled. External data interfaces are indicated with arrows entering or leaving the diagram at the corresponding adapter.

As it can be seen, the persistence layer consists of three components. The XML-Persistence takes care to persist everything which is necessary to represent the state of the mission plan and the MPT itself. This includes classes and methods to save and load the mission plan, all procedures, resources including their states, and general settings of the MPT as XML-files. The decision to implement this persistence component using the XML standard was anticipated by MPT-D-0020 (cf. 4.3) in response to the requirement to store all of the MTP’s user data in a human readable format to prevent loss of data of the mission model and the mission plan, even if the MPT cannot be executed anymore anytime in the future for any reason. In contrast to other human readable formats, the XML-files are also very easy to programatically parse and assemble as thanks to the standardisation reliable libraries exist for numerous platforms and programming languages. The two components TC Database and TM Definition mainly implement the interface to the corresponding mission dependent databases. They are nevertheless modelled as own components and not only external read-only interfaces, as they may also include functionalities to model constraints on the abstraction level of telecommands (cf. 4.1.2), or the translation between actual telemetry data and resources modelled for the MPT (cf. 5.4).

The model consists of the MissionPlan, which mainly holds and manages the timeline entries and provides the full list or subsets of it to other components, and takes care of general information about the mission plan, like the planning horizon\(^1\), whether the current plan is valid etc. The Procedure component unites all functionalities which are related to the modelling of procedures, including the algorithms for the procedure validation functionality. It provides the list of steps and constraints of a procedure as interface to other components. The Resource component models the resources, which not only includes the definition of the resources themselves, but also the algorithm to propagate them and check for conflicts according to the effective constraints and general information of the mission plan. Resources which are too complex to be modelled within the MPT, like thermal models, can be plugged in as external resources.

The Sequence of Events Monitor component (cf. 4.3.2) simply displays the complete list of steps to be conducted and originating from the different timeline entries and provides functionalities to configure the monitor (i.e. filter or sort the list etc.). It is hence part of the view layer. The information to be presented by the SoE monitor is

\(^1\)The planning horizon is the timeframe which is currently regarded by the MPT. Depending on the mission it commonly ranges from few weeks in the past to a couple of weeks in the future.
provided by the control component *SoE Generator*, which assembles the sequence of events given the timeline entries and general information from the mission plan and the steps of the underlying procedures, as well as user settings (like filters, sorting criteria) provided through the dialog. The sequence of events is additionally provided as external data interface according to 4.3.1.

Another component of the view layer is the *Planning Timeline* (cf. 4.3.2) which graphically and textually represents the currently scheduled timeline entries, as well as the course of time of resources and conflicts. Furthermore, it provides interfaces to change the current mission plan. The communication to the model is brokered by the *Planning Controller* which is part of the control layer. It combines all information necessary for the scheduling of timeline entries and calculates whether a scheduling request of the user can actually be performed. In a more advanced version also algorithms for automatic conflict resolution could be part of this component.

The *Procedure Editor* component which is also part of the view layer provides the user interface to model procedures (cf. 4.3.2). According to the MVC paradigm, the *Procedure Editor Controller* serves as agent between the procedure component of the model layer and the procedure editor of the view layer.

Where applicable, especially towards the persistence components, the interfaces between the components are handled by Java interfaces to facilitate future changes of the implementing classes. Please note, that besides the discussion of the subject specific classes in 6.3, the implications of the Eclipse Rich Client Platform (cf. 6.4.3), and explanations regarding the actual implementation (cf. 7), the specific design of the components cannot be elaborated in detail as it would go beyond the scope of this thesis. Interested readers may refer to the technical documentation (cf. 7.6).

### 6.3 Detailed Design of the Subject Specific Classes

Based on the model of the subject specific classes, which was already introduced during the object oriented analysis (cf. 4.2), and the general system architecture of the previous section (cf. 6.2), the design of the most relevant classes will be discussed in the following subsections. Furthermore, two important sets of auxiliary classes are discussed as well.

#### 6.3.1 Mission Plan

The *MissionPlan* class is the core class of the mission plan component and ultimately the container of all relevant data of the current mission plan. To guarantee that all other components which use its interfaces always access the most current state, it is modelled according to the singleton design pattern. As fields it holds

- a reference to the *TimeZero* object (cf. 6.3.5),
- two final *RelativeTimepoint* objects (cf. 6.3.5) which represent the start and end of the planning horizon,
6.3. Detailed Design of the Subject Specific Classes

- two further final RelativeTimepoint objects which represent the date until when the mission plan has already been exported and until when the export is up-to-date,
- a string representing the status of the exported mission plan,
- objects which represent a local copy of the telecommand database,
- the list of TimelineEntry which represents the currently scheduled timeline entries,
- a set of Subsystems which represents the organisational units which are relevant for the mission planning,
- a boolean flag dirty, indicating whether the plan had been changed since the last saving,
- the set of ISchedulingTimelines which currently are subscribed to the mission plan to be notified when the plan changes.

Besides getter and setter methods for all those fields, there are methods to add, update, and remove a TimelineEntry. Regarding the list of TimelineEntry there is a couple of methods which filter the list with either a Procedure or a Subsystem and the immediate field of the timeline entry as search criteria. Similarly, there are methods to add and remove a Subsystem or a subscribing ISchedulingTimeline to the corresponding sets and a method which returns the Subsystem given its identifying name. The methods save and saveAs directly delegate to the persistence component and set the dirty flag to false. The private method updateTimelines notifies the set of subscribing ISchedulingTimeline whenever the list of TimelineEntry is changed. An auxiliary methods checks for a Procedure whether it is used in the current mission plan. Finally, there are methods which assemble the sequence of events either for the whole planning horizon or a specified start and end timepoint and containing timeline entries either containing the complete list, only ones marked as immediate, or only ones marked as time tagged. Those methods, which return a time-ordered list of SpecificStep, would actually be part of the SoE generator component if the system architecture discussed in 6.2 was strictly obeyed. They were moved to this class as thanks to the powerful capabilities of the Java Beans Databinding framework (cf. 7.2.1) the SoE generator component can be reduced so much, that no special classes are needed and the the SoE generator component can so to say be regarded as "virtual component".

6.3.2 Timeline Entry

As already defined in 4.2 a TimelineEntry is a specific manifestation of a Procedure. Therefore the TimelineEntry holds two RelativeTimepoint objects, which represent the actual start and end date of the timeline entry and commonly are directly based on the TimeZero object (cf. 6.3.3), and a reference to the underlying
Procedure. From a subject specific point of view it furthermore contains a flag, indicating whether the timeline entry shall be conducted in an immediate or time-tagged manner and the set of SpecificProcedureParameter, i.e. the initialisation of the ProcedureParameters declared in the Procedure (refer to 6.3.6 for a full understanding of the design of parameters related to procedures and timeline entries). Besides those, the TimelineEntry class has a few additional fields which are necessary or useful from a technical point of view, but of no relevance for the business logic and therefore omitted here.

Most of the methods provided by the class are setters and getters for all those fields and methods to modify the set of SpecificProcedureParameters. However, the most important method is getGroundedParameterizedSteps. Based on the start and end date of the timeline entry and the parameter initialisation, it "translates" the Steps of the underlying Procedure into a list of SpecificSteps which all reference to an absolute point in time and have the parameters replaced by their initialized values. This method is called by the MissionPlan when assembling the SoE for each timeline entry and thus could also be regarded as part of the "virtual" SoE generator component. Furthermore, the TimelineEntry class includes some auxiliary methods to find the SpecificProcedureParameter object given its identifying name and methods to provide strings which are displayed by the GUI to represent the timeline entry.

6.3.3 Procedure

The Procedure class is the core of the procedure component and more or less the heart of the whole MPT. The class is accordingly complex and consists of the following fields.

- final RelativeTimepoint objects, which remain ungrounded, representing the start and end of the procedure,
- integers representing the minimum and maximum duration of the procedure,
- a string describing the purpose and aims of the procedure in natural language,
- a set of ProcedureParameter holding all parameters used in the procedure (cf. 6.3.6),
- flags which indicate whether the procedure shall be conducted in an immediate or time-tagged manner, and whether it contains autonomous parts or is to be scheduled purely classically (cf. 5),
- a set of Subsystems which represents the organisational units which will be occupied by the procedure,
- strings representing the preconditions, pass criteria, and fail criteria which apply to the procedure,
• the ordered list of Step holding all steps to be conducted throughout the procedure,

• a string representing the path of the procedure (i.e. the procedure's name and location in the procedure repository),

• some members of technical relevance, omitted here.

The methods provided by the class are basically all related to the access of the fields, i.e. getters and setters and in the case of the more complex datatypes more advanced methods, e.g. to access a Step at a specific index of the list. Besides helper methods, which determine if a parameter, identified by its name, is already defined for the procedure, the class provides no further algorithmic methods.

6.3.4 Step

A step is the atomic unit of the mission plan. The corresponding Step class holds a reference to the Procedure the step is part of, a RelativeTimepoint (cf. 6.3.5) representing the start of the step. This timepoint can either be based on the start or end object of the procedure. Furthermore, it features two strings describing the action related to the step and its predicted result, and a Parametrizable<Integer> (cf. 6.3.6) representing the duration of the step. Via an optional field (i.e. it may be null) a telecommand (i.e. SpecificCommand) can be assigned to the step. Similarly, to the other classes it also consists of some members which are only of technical importance and thus omitted here. The class solely provides getter and setter methods for all its fields, but no further logic.

The SpecificStep class is the equivalent of Step when the complete SoE is assembled. Thus, it basically features the same member variables as Step. The only difference is, that the start is required to be an absolute point in time, the duration no longer a Parametrizable<Integer> but a pure integer and that the step inherits the flag, whether it shall be conducted in an immediate or time-tagged manner from the corresponding timeline entry.

6.3.5 Timepoint

For proper mission planning a consistent timing of all events is essential. Because firstly a time format broadly used in the space sector, the mission elapsed time (MET) is not supported by standard time libraries, and secondly those libraries heavily depend on the system time and its time zone, while it might be necessary for the planning engineer to use time zones different from his system's time, it was decided to mostly avoid those problems and develop some basic but yet powerful auxiliary classes for the timing.

The main idea of the developed concept is, that there is one (single) fixed point in time which is associated with a UTC date and time. All other timepoints are eventually expressed relative to this timepoint. While it is absolutely required, that
every timepoint can be related to a UTC date and time when the SoE is eventually
generated, during the planning process "ungrounded" timepoints specifying a relative
relation between several points in time (like "5 s after the start of the procedure") are
required. Both shall be possible and it shall be easy to "hook" and "unhook" those
possibly complex relative time dependencies to a fixed point in time. For instance,
when a timeline entry gets scheduled for a fixed point in time, suddenly "5 s after the
start of the procedure" can be translated into a fixed point in time. This shall be
possible as easily as possible.

The abstract Timepoint class is the starting point of the concept. It defines the four
abstract methods getBase, getTimeToBase, isGrounded, and getUTC. The method
getBase shall return the timepoint, this timepoint is based on, following a possible
long linear chain of dependencies. getTimeToBase returns the number of seconds the
calling timepoint is offset to its base. isGrounded tests whether the timepoint can
be related to a fixed point in time and getUTC converts (if possible) the timepoint
into a ZonedDateTime in UTC and thus provides the bridge between the own timing
concept and the standard libraries. The class furthermore provides the method
equals(Timepoint t), which returns true iff the compared Timepoints have the
same base and the same time to this base. Furthermore, it provides two methods
which provide strings of the timepoint, if a conversion into a UTC timestamp is
possible.

This abstract class is extended by the three classes TimeZero, RelativeTimepoint,
and UngroundedTimepoint.

TimeZero represents the single fixed point in time and is thus a singleton. It has a
field which stores the UTC date and time of this timepoint as ZonedDateTime object.
Being requested its base, the class returns this, accordingly 0 when asked for the
time to its base and as it is directly associated with a UTC date time always returns
true, when tested whether it is grounded. Furthermore, it provides methods to set
the UTC date time, it is associated with, and static helper methods to calculate and
format the MET for a given other UTC ZonedDateTime.

The counterpart of TimeZero for ungrounded timepoints or for the end of long
ungrounded relative time dependencies is UngroundedTimepoint. Thus it always re-
turns this when asked for the base, returns false when tested whether it is grounded,
returns 0 for its time to base, and null when the UTC ZonedDateTime object is re-
quested.

Finally, a RelativeTimepoint is the flexible (and most used class) to build complex
time dependencies. It has another Timepoint (either of the three extending types) as
base and a Parametrizable<Long> (cf. 6.3.6) as offset in seconds to this base. If no
base is specified in the constructor it is set to a new UngroundedTimepoint. When
the base is requested, it recursively traverses the chain of bases until it ends either at
a UngroundedTimepoint or TimeZero. Similarly, the methods isGrounded, getUTC,
and getTimeToBase work recursively, while the latter two add the own offset to the
result provided by the recursive call. Unlike the other two classes, it provides setter
methods to modify the offset and base.
Thus, all timing within the MPT is modelled as seconds to the TimeZero singleton, and only where necessary converted into ZonedDateTime objects. Furthermore, complex time dependencies and ungrounded points in time can be modelled. Also hooking and unhooking the RelativeTimepoints is really easy. For instance, when a timeline entry is scheduled, its start and end are simply set to be based on TimeZero with the correct offset. If now the SoE shall be generated, the start and end of the underlying procedure have only to be temporarily based on the timeline entry’s start and end (with offset 0) and a simple call of `getStart().getUTC()` for every step automatically returns the correct UTC time date.

6.3.6 Parametrization of Procedures

When talking about procedures it was silently neglected up to this point, that for a proper MPT it is essential that procedures must be parametrizable up to some extend. To convince yourself about this, simply imagine a procedure for an earth observation satellite which shall take a snapshot of a target on ground. The procedure is rather simple, e.g. following those basic steps

1. Point camera to target
2. Switch on camera
3. Take snapshot
4. Switch off camera

But already the first step is the culprit. If no parameters were allowed or possible in the scope of a procedure, the procedure would have to be modelled for each and every target having the target hard-coded into the procedure – an obviously tedious, if not endless task, further considerations like the maintainability of the procedures not even mentioned. It would be much better (and thus eventually safer for the mission), if the target was a parameter (i.e. represented by the attitude angles required to point to it) which could be used as variable or placeholder within the procedure definition and the actual values were only to be defined, when a timeline entry based on the procedure was actually scheduled.

To implement this feature three classes are designed. The generic class `Parametrizable<T>` is a container for objects which can be parametrized. It features a string as possible placeholder symbol, an object of type T as value and a final object of type T as default value. It provides getter and setter methods both for the symbol and the value. In contrast to common setters they set the respectively other field to null, as a `Parametrizable<T>` can only either be a placeholder or have a value and be treated like normal variable of type T from the outside. The getters throw accordingly exceptions if the program tries to access the value if the `Parametrizable<T>` is to be treated as parameter and vice versa. This type is used instead of the type T in the implementation for any variables which shall be parametrizable. This is for
instance the case for the duration of a step (Parametrizable<Integer>), the offset of relative timepoints (Parametrizable<Long>) or the SpecificParameters of a SpecificCommand (Parametrizable<String>). When modelling a procedure, the systems engineer can thus decide whether to provide a constant value (which does not change for different timeline entries with the same underlying procedure) or a placeholder symbol. To enable proper parsing, a placeholder symbol has to start with an @ and consist of at least one further character.

Similarly to programming languages, where variables have to be declared before they can be used, the systems engineer has to add a ProcedureParameter to the procedure. A ProcedureParameter consists of a symbol (string), a type reflecting whether the parameter is a placeholder for an integer, a floating point number or a string, a descriptive string and a string for a unit. Both the descriptive string and the unit have the sole purpose to inform the planning engineer about the physical meaning of the placeholder.

When a timeline entry with an underlying procedure which contains parameters is to be scheduled, the planning engineer is asked to provide a value of the correct type for each ProcedureParameter. This value is stored in a SpecificProcedureParameter object, which extends the ProcedureParameter and inherits all other information from it. This step can be compared to the variable initialisation in programming languages.

Finally, when the sequence of events is to be assembled, the SoE generator checks for every step of the procedure, which of its Parametrizable<T> elements are placeholder variables. It then looks up (with the symbol as identifier) the corresponding value in the set of SpecificProcedureParameters of the timeline entry and replaces the placeholder with the actual value when creating a SpecificStep.

To illustrate this concept, consider the example of figure 6.2. The procedure genericWheelTest shall check-out reaction wheels. As this procedure is to be conducted separately for every reaction wheel on board, the ProcedureParameter @axis of type INT is defined which represents the axis of the wheel (cf. figure 6.2a). This placeholder is then used (among others) as parameter of the telecommand which switches on the correct reaction wheel (cf. figure 6.2b). When scheduling a timeline entry of genericWheelTest, the planning engineer assigns the value 1 (which could for instance represent the satellites x-Axis; cf. figure 6.2c). When the sequence of events is assembled, the placeholder @axis is replaced by its assigned value 1 (cf. figure 6.2d).
6.4. Eclipse Rich Client Platform

6.4.1 Historic Overview

The Eclipse Rich Client Platform (RCP) is one of the flagship projects of the Eclipse Foundation. It was originally not planned as stand-alone project but emerged from the development of the famous Java IDE Eclipse. 2001 IBM presented Eclipse as the successor of its former Java IDE VisualAge. While VisualAge was a monolithic application with the main disadvantage of difficulties to integrate external tools, a radical design change was introduced with Eclipse. Instead of a monolithic application, the
IDE consists of a set of plug-ins, even for core functionalities. Thus it can easily be extended with external tools, but also the core plug-ins can easily replaced by new ones\textsuperscript{2}. With the introduction of the Eclipse IDE, IBM opened it for the open-source community and the Eclipse Foundation was founded. Only in the course of time, it was recognized that the innovative design concepts of the IDE could advantageously be ported to other applications than the IDE as well – the Eclipse Rich Client Platform was born and huge efforts were made to separate the platform components from the actual IDE. In 2004 with the release of the Eclipse IDE version 3.0 the Rich Client Platform, which is commonly called RCP 3 as based on the Eclipse IDE 3.x, was introduced. A new version of the RCP, called RCP 4 as based on the API of the Eclipse IDE version 4.x, was announced in 2008. The new version features several improvements which make the platform even more attractive, but is also backwards compatible to RCP 3 (cf. \cite{Steppan, 2015, pp. 13-16}).

Throughout the following two subsections it shall be argued, why the MPT was developed as RCP 4 application, and the implications of this huge design decision on the overall design are outlined.

6.4.2 Reasons for RCP Development

The main reason to use the RCP as a starting point for the development of the MPT is indeed the modularity. As it is obvious already from the analysis (cf. \ref{4}) and the system architecture (cf. \ref{6.2}) a modular design is advantageous for the MPT. Even regarding the minimal set of functionalities and use cases, the distinction between the system engineer, the planning engineer, and the operator as the main users make it favourable to separate the development especially of the GUI components into different modules. Furthermore, the vision of a powerful MPT featuring also advanced functionalities like the resource control, or perhaps in a long term also automated scheduling, while those visions can obviously to no extend be reached in the scope of a master’s project, is another argument to keep the MPT as modular and open for extensions as possible. Thirdly, regarding especially components of the persistence layer (cf. \ref{6.2}) it is favourable for the future usage of the MPT also for other missions to facilitate plugging in new components, i.e. to connect to different telecommand databases.

Besides the conceptual advantages of the modularity, the experience with other RCP applications developed at the Professorship for Space Technology at the Chair of Aerospace Information Technology of the University of Würzburg, like the telecommand client (cf. \cite{Beylin, 2014}), shows that the modularity and the usage of the platform has also a positive impact on the stability of the application through two major features. Firstly, advanced functions for the GUI design (e.g. different perspectives) are natively supported by the platform and existing plug-ins, and thus the risks own implementations always bear can be avoided. Secondly, the fact, that the platform

\textsuperscript{2}This leads to the nearly philosophical idiom among Eclipse developers and contributors: "Everything is a plug-in."
catches errors of single modules (e.g. due to erroneous user input, which was mistakenly not regarded by the error handling of the module) while the application as a whole remains stable and thus can be safely shut-down without loss of data. Additionally, the RCP strongly encourages the MVC paradigm (cf. \cite{6.1}) and facilitates the (cross-platform) deployment of the final software product.

The main disadvantage to develop the MPT as RCP application is the increased learning effort to develop the application on top of the platform. Although \cite{Steppan, 2015} and \cite{Vogel, 2015} provide profound introductions to RCP development, they are to no extent complete. Thus, many problems were encountered during implementation which made it necessary to perform time-consuming tasks to narrow down the "magic problem", do research in developer forums etc. or solve the problem with systematic trials (and errors). This learning effort and the time consumed to solve the encountered problems was heavily underestimated during the early stages of the project and are basically the reasons, why none of the secondary objectives were achieved (cf. \cite{3.2.2} and \cite{8.2.6}). However, as this did not affect the primary objectives, it has no impact on the overall success of the project (cf. \cite{8.2.6} and the benefits of a modular application, which can easily be extended with any further advanced functionality, clearly justifies the extra learning effort for the usage of the RCP.

### 6.4.3 Design Implications

The design of the model and persistence is not affected by the utilisation of the Eclipse Rich Client Platform. This is not the case for the control and view components. The hub of every RCP application is the Application Model. This is an XML-file, which is parsed by the RCP core components at the application start-up and defines which other Java classes are to be loaded and instantiated. To do so, the developer can define so called parts, which are the atomic GUI modules, and their general accommodation within the application window(s). Those parts are to be linked with specific classes, which implement the parts' appearance and behaviour. The classes related to a part must provide a standard constructor without any arguments, which commonly only performs very basic tasks. Most of the work, i.e. the assembly of the GUI, is implemented in a method which is annotated with \@PostConstruct and called by the platform right after the constructor while instantiating a part. The class may feature other methods annotated with \@PreDestroy, which is called right before the part is destructed, or \@Persist which is necessary if any data of the part shall be persisted and which is called if saving the data is invoked by the platform. Besides those methods, the class associated with a part may of course implement any methods which are necessary for its functionalities. Subclassing (and delegating to the annotated methods of the super-class) is supported.

Besides parts, also menus, commands and handlers can be defined in the Application Model. Menus can either be attached to windows or parts. Not going into details, a menu item is associated with a command and the command with a handler. The handler is then, similarly to the parts, associated with a class, which implements
its behaviour. Also those special classes have to provide a standard constructor without arguments and an annotated method. This method, annotated with @Execute, is called when the handler is executed, e.g. the corresponding menu item clicked by the user. Another optional method, annotated with @CanExecute and returning a boolean, decides whether the handler can be executed or not, e.g. whether the corresponding menu item shall be enabled or disabled.

The experienced reader will probably wonder, how it is possible to pass necessary (model) data to the part and handler classes if they are instantiated by the platform through the standard constructors without arguments. This is possible in four different ways. The most elegant, efficient, and broadly used in RCP 4 applications is Dependency Injection (DI). The platform has control over the whole context of the application. With special annotations (like @Inject or @Named) the platform can be requested to provide parts of this context to the corresponding scope (i.e. either as a parameter of a method, or as a member variable of a class). By default this includes information of the platform itself, i.e. which parts are currently instantiated, which window is in the foreground etc., but own (model) data can explicitly be added to the context and can thus be provided to other classes through DI. The second possibility is restricted to cases, where the required data is part of the data taken care of by the persistence components. The corresponding part or handler class which requires some of this data could simply delegate the request to the persistence components in its constructor. The third method requires that the requested data to be provided by a singleton class. The requesting class can simply access the singleton instance by the static method in its constructor and then access the data through the instance. The fourth method applies only, if the corresponding class is not automatically instantiated by the platform but explicitly by the developer in the scope of any other source-code. The corresponding class could simply provide additional methods (i.e. a second constructor with the corresponding arguments, or a specific setter) which pass the required data to the class.

All those characteristics of the RCP of course have to be regarded throughout the design and implementation of the MPT.
CHAPTER 7

Implementation

This chapter shall give a brief overview over the implementation of the MPT. It mainly aims at future developers to gain a broader understanding of the actual software product before digging deeper into the technical documentation and source code. Nevertheless, it is also valuable for users to understand the functionalities of the MPT. A more detailed description for users, i.e. a user handbook, is provided in appendix B. As the actual structure of the implementation is highly influenced by the method of software development, which was applied throughout the project, this is shortly summarized in the first section. Secondly, this chapter provides an introduction to the used external technologies and libraries. Thirdly, the package structure, where to find which functionalities in the sources, is sketched and explained. The fourth section presents the actual GUI of the final software product, which mainly complies to the considerations of the analysis (cf. 4.3.2) but differs in many small details. The chapter concludes with short remarks on the documentation of the source code and the deployment of the final software product.

7.1 Applied Method of Software Development

After the precise definition of the objectives and requirements and the thorough object oriented analysis, which were conducted in a classical static (yet of course iterative) manner which is typical for space related projects, the working methods were changed to more agile ones, as they are by today very common for software projects, for the design and implementation. This was easily possible, as unlike other projects, which would require also personnel restructuring in the case of such a change of methodology, the project was run like most master’s projects as a "one-man-show". The step towards more agile methods means, that instead on spending endless hours on the theoretically best overall design, one starts with the implementation of small parts until a working version (as measured by the corresponding requirements) is achieved. If the design of this version, when used later in other contexts, turns out to be problematic, it is restructured and refactored. This leads to many often
undocumented iterations of the sources, which is supported by version control tools (SVN) and refactoring tools of the used development environments. The disadvantage of this method is, that the quality of the code might be worse (when design pitfalls remain undiscovered), but has the big advantage of executable and working software at any time of the project. Especially for the quite short project lifecycle of six months, this was absolutely beneficial.

7.2 Used External Technologies and Libraries

7.2.1 Standard Widget Toolkit & JFace

The Standard Widget Toolkit (SWT) is, besides others like the Abstract Widget Toolkit (AWT) or Swing, one of the most popular user interface toolkits for Java. It is broadly used for RCP applications and provides classes for all basic components of user interfaces (like buttons, text fields, simple tables). The SWT directly delegates the drawing of the UI components to the operating system wherever possible and emulations are only provided for widgets the corresponding operating system does not provide. This makes it less platform independent than AWT or Swing, which is uncommon for Java applications, but guarantees a proper "look and feel" as of a native application. As RCP applications anyway need to be deployed separately for every platform, the increased platform dependency is no major drawback, but the native "look and feel" a great benefit for the users.

While the SWT itself only provides the basic components, where the close coupling to the operating system is favourable, more advanced components (tables, trees etc.) are abstracted into the JFace library which is a pure Java library and implements interfaces to the SWT. Besides the more complex UI components, JFace provides powerful classes for the databinding, which facilitate the implementation of the MVC paradigm (cf. [6.1]) a lot.

For so called Java Beans, i.e. classes which encapsulate several other objects as member variables, provide a zero-argument constructor, and getter and setter methods for all member variables, JFace provides classes, to "bind" the Java Beans class (and its member variables) directly to the UI widgets. For instance, in case of a text field this would mean (depending on the configuration of the databinding), that JFace loads the value of the bound member variable and sets it as the value of the text field on creation, and vice versa updates the member variable whenever the user changes the value of the text field.

To do so, basically two things have to be regarded during implementation. Firstly, the model has to inform the GUI components about any changes via a java.beans.PropertyChangeSupport object. Commonly, this is done by an abstract class AbstractModelObject which is sub-classed by all relevant model classes, and which delegates all methods associated with the information of the other GUI components to a private PropertyChangeSupport object. The relevant method is then firePropertyChange(). Following the Java Beans naming conventions, a model
7.2. Used External Technologies and Libraries

Listing 7.1: Example for model class prepared for JFace databindings.

```java
public class ConcreteModelObject extends AbstractModelObject {
    private Type1 modelMember1;

    public Type1 getModelMember1() {
        return this.modelMember1;
    }

    public void setModelMember1(Type1 arg0) {
        firePropertyChange("modelMember1", this.modelMember1, this.modelMember1 = arg0);
    }
}
```

Listing 7.2: Example for databinding initialisation.

```java
ConcreteModelObject myBean = new ConcreteModelObject(); // the model object
Text myText = new Text(parent, SWT.NONE); // the GUI widget
DataBindingContext myBindingContext = new DataBindingContext();

IObservableValue observeWidget = WidgetProperties.text(SWT.Modify).
    .observe(myText);
IObservableValue observeBean = BeanProperties.value("modelMember1").observe(myBean);

myBindingContext.bind(observeWidget, observeBean, null, null);
```

class can look like the example in listing 7.1. Secondly, the databinding, between a Java Bean class and a GUI widget has to be initialized. Two objects which implement the interface IObervableValue, which can either be the widget or the member of the Java Bean, can be bound to a DataBindingContext. Listing 7.2 shows a minimal example. The initialisation of the databindings is strongly supported by a wizard of the IDE, i.e. the widgets and Java Bean members can be selected and their binding specified on dialogs and the corresponding code for the initialisation is automatically generated.

With those powerful capabilities most of the control components (cf. 6.2) condense basically to the proper preparation of the model classes and the initialisation of the databindings which is done in the classes of the corresponding view components.
### 7.2.2 SWTChart

SWTChart is a small yet handy and well-documented open-source library, which is based on SWT, to draw charts and plots. It is used to draw the timelines of the planning timeline component (cf. §6.2). For further information about the library, please refer to the documentation (cf. [swtchart project, 2016]).

### 7.2.3 Java Architecture for XML Binding

The Java Architecture for XML Binding (JAXB) is an API which allows direct mapping between XML-files and Java Objects and vice versa. Instead of assembling and parsing the XML-files which shall be used to persist all procedures and the current mission plan (cf. MPT-D-0020 in A.3) "manually" in the persistence component (cf. §6.2) this can automatically be done by JAXB, assuming that the classes which shall be mapped to XML-files are Java Beans and that some special annotations are given in their source code. Classes, which shall be the root element of the XML-file, have to be annotated with @XmlRootElement (cf. listing 7.3) and provide a zero-argument constructor. They can then be automatically converted to XML-files (cf. listing 7.4 result in listing 7.5) and XML-files back to the Java object (cf. listing 7.6). To do so, JAXB scans the class for public member variables or ones with public getter and setter methods, which is done recursively for non-trivial types of member variables. Some other annotations can be used, to assign names of the XML elements different to the member variables' names, to skip member variables, or to put member variables as XML attributes instead of XML elements. For a broader introduction for the usage refer to [Ullenboom, 2014, ch. 19.4].

With this powerful API the implementation effort of the XML-persistence component (cf. §6.2) reduces to properly annotating the classes to be persisted, implementing the marshalling and unmarshalling snippets, and taking care of very few special cases. An example of such a special case is the parent field of a Step which refers to the Procedure the step is part of. This field is annotated to be skipped by JAXB as the information is actually already reflected as the step is part of the procedure's stepList and as otherwise JAXB would run into a endless recursion, trying to assemble the whole XML structure of the referenced parent procedure, including the step, which would refer to the procedure again, and so on. Thus, this field has to be manually set when a procedure is loaded by the persistence component, as otherwise it would remain null causing null pointer exceptions when accessed.
Listing 7.3: Example for model class prepared for JAXB.

```java
@XmlElement(name="myObject")
public class ConcreteModelObject extends AbstractModelObject {
    @XmlAttribute(name="mem1")
    private Type1 modelMember1;
    public Type1 modelMember2;
    @XmlElement(name="newName")
    public Type1 modelMember3;
    @XmlTransient
    public Type1 modelMember4;
    public Type1 getModelMember1() {
        return this.modelMember1;
    }
    public void setModelMember1(Type1 arg0) {
        firePropertyChange("modelMember1", this.modelMember1, this.modelMember1 = arg0);
    }
}
```

Listing 7.4: JAXB marshalling ConcreteModelObject (cf. 7.3).

```java
public void save(String path, ConcreteModelObject object) throws IOException {
    JAXBContext context;
    try {
        Files.createDirectories(Paths.get(path));
        context = JAXBContext.newInstance(ConcreteModelObject.class);
        Marshaller m = context.createMarshaller();
        m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
        m.setProperty(Marshaller.JAXB_ENCODING, "UTF-8");
        m.marshal(object, Paths.get(path).toFile());
    } catch (JAXBException e) {
        throw new IOException(e);
    }
}
```
Listing 7.5: JAXB resulting XML-file for a ConcreteModelObject (cf. [7.4]).

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<myObject mem1="...">
  <modelMember2>
    ...
  </modelMember2>
  <newName>
    ...
  </newName>
</myObject>
```

Listing 7.6: JAXB unmarshalling ConcreteModelObject (cf. [7.5]).

```java
public ConcreteModelObject load(String path) throws IOException {
    ConcreteModelObject result;

    InputStream inputStream = new FileInputStream(path);
    Reader reader = new InputStreamReader(inputStream, "UTF-8");
    JAXBContext context;
    try {
        context = JAXBContext.newInstance(ConcreteModelObject.class);
        Unmarshaller um = context.createUnmarshaller();
        result = (ConcreteModelObject) um.unmarshal(reader);
    } catch (JAXBException e) {
        throw new IOException(e);
    } finally {
        reader.close();
    }
    return result;
}
```

### 7.2.4 Java Database Connectivity

The Java Database Connectivity (JDBC) is an API which is part of the Java Standard Edition platform and is de-facto standard for Java to connect to relational databases. As the telecommand database is a SQL database, the TC database component (cf. [6.2]) takes advantage of the API to connect to the telecommand database and fetch a local copy to support the modelling of procedures. Because this functionality is completely adapted from the existing telecommand client which also connects to the telecommand database, neither the working principles of JDBC nor the database structure of the telecommand database of SONATE are elaborated here. Please refer to [Beylin, 2014] and [Ullenboom, 2014, ch. 18] for more details on the SONATE telecommand database and general information about JDBC respectively.
7.3 Package Structure

To facilitate the understanding for future developers, i.e. where to look for what, the structure of the Java packages of the final software product and naming conventions for classes are briefly outlined here.

7.3.1 Model

The package sonate.mpt.rcp.model contains all classes of the model components (cf. 6.2). Besides the classes already discussed in detail in 6.3 this includes the AbstractModelObject, as well as classes for the clocks to be displayed and the class for the local copy of the telecommand database and some auxiliary classes (like ClockTreeParent or TimelineEntryComparator) which are of technical importance only.

7.3.2 Parts

The package sonate.mpt.rcp.parts contains most of the classes related to the view components (cf. 6.2), namely all classes related to RCP parts. The name of classes which are to be directly linked to a part in the application model (cf. 6.4.3) end with the keyword View. All others are auxiliary classes of technical relevance. To further structure the package and make things easier to find, two subpackages are introduced. sonate.mpt.rcp.parts.timelines contains all classes associated with the planning timeline component (cf. 6.2) and sonate.mpt.rcp.parts.panels contains all classes related to any information panels.

7.3.3 Dialogs

The package sonate.mpt.rcp.parts contains basically the rest of the view components, i.e. the classes needed for complex dialog windows. This includes the preferences dialog, the dialog to export the SoE, the dialog to set the planning horizon, the wizard to create a new timeline entry, the dialog to change the details of an existing timeline entry, and the dialog to save a new procedure. The names of all those classes end with Dialog and all classes extend the JFace TitleAreaDialog.

7.3.4 Handlers & Control

The package sonate.mpt.rcp.handlers contains most of the control components (cf. 6.2) which is not taken care of by the JFace databindings (cf. 7.2.1). All classes are handlers (suffix Handler of the class name) which are registered to the RCP application model (cf. 6.4.3), i.e. take care of the activities to be conducted when the user clicks any button or menu item.

The package sonate.mpt.rcp.control contains all remainders of the control components, which are neither handlers nor covered by JFace databindings. Currently
this is only one class, which runs a separate thread which invokes the update of all registered IclockedParts every second, and another one to control the locking and unlocking of the UI components (cf. MPT-F-0310 in A.1).

7.3.5 Persistence

The package sonate.mpt.rcp.persistence contains all classes related to the persistence components (cf. 6.2). Java interfaces which are defined to increase the portability are indicated by the name-prefix I, e. g. IMissionPlanPersistance. The implementing classes feature a name-suffix representing the underlying form of persistence (e. g. XML for XML-files, CSV for comma separated files, or SQL for SQL-databases). While those classes directly take care of the persistence of the specific element, in some cases model provider classes are implemented as intermediate class towards the other components. This is for instance the case for the procedure persistence. While IProcedurePersistence (and the implementing class ProcedurePersistXML) handles the saving and loading of a specific procedure or to scan the whole repository, the ProcedureModelProvider builds and provides the tree used for categorized presentation of the defined procedures in various UI components.

7.3.6 Other

The package sonate.mpt.rcp.ui.controls includes more complex UI elements, which are not related to RCP parts. Currently, this is only the case for the perspective toolbar (cf. B.1), which is adapted from [Steppan, 2015. ch. 23.14]. org.eclipse.wb.swt contains auxiliary classes for SWT which were automatically generated by the IDE. The root package sonate.mpt.rcp contains the remaining miscellaneous classes, mainly the Timepoint classes (cf. 6.3.5) and Parametrizable (cf. 6.3.6).

7.4 Final Graphical User Interface

Throughout the agile implementation, some details of the concept for the GUI as developed during the object oriented analysis (cf. 4.3.2) turned out to be impractical for the actual usage. Therefore, this section shall briefly present the final design of the GUI and discuss the improvements compared to the pencil sketched prototypes presented before. For a more comprehensive usage of the MPT, please refer to the user handbook in appendix B.

7.4.1 General

The final design of the GUI features three improvements, which were not directly regarded in the analysis, but are valuable for the user. Firstly, the deployed version (cf. 7.5) comes along with a distinct icon on the task bar. Thus, the user can easily identify the MPT among other running applications. Secondly, the title bar of the
MPT can be set by the user via the preferences. This is especially valuable, for possible multi-mission scenarios where different instances of the MPT are installed for different missions. The customized title bar could then help the user to identify the instance at first glance. Thirdly, thanks to available standard RCP plug-ins the GUI is highly flexible. All parts can be changed in size, minimized, maximized or even detached from the main window, e.g. to be displayed on a second screen. Thus, the user can flexibly configure the appearance of the MPT to his specific needs.

While the basic structure, three different perspectives for the three different main users, stays the same, each perspective features improvements. Hence, the discussion of those is presented separately for each perspective.

### 7.4.2 Planning Perspective

The design of the planning perspective as seen in figure 7.1 apparently was not changed much compared to the first prototype (cf. 4.3.2). One of the major changes is the top panel. It not only displays three clocks (UTC, local time, and MET) rather than one as in the prototype, but displays also more general information about the mission plan. The lock/unlock button was moved to a menu entry in order to be accessible also from the other perspectives. On the left hand side, the selection of the displayed timelines was realized with tabs rather than different tree nodes mainly for technical reasons. However, while "Resources" is missing as the resource control functions were not implemented, a new category "Procedures" was introduced. Through this selection, the planning engineer can additionally inspect the mission plan categorized by the procedures, not only the occupied organisational units. The conflict console is also missing due to the fact that the resource control remains unimplemented. The details view is moved to the left to keep as much space as possible for the planning timelines and is a pure display, i.e. has no controls to change the selected timeline entry, as those can be accessed directly through a context menu attached to the timeline entry on the timeline. Furthermore, the timelines can be zoomed and can either be coupled to the clocks and thus scrolled and zoomed synchronously or remain uncoupled with an own time axis and scrolled and zoomed independently. Depending on the monitor resolution, some (very short) timeline entries would disappear at low zoom levels (showing a large time range). To avoid this the user can set a minimal width in the preferences. As additional hint the label at the beginning of the timeline indicates by its background color, whether there is a timeline entry in the currently shown range.
7.4.3 Procedure Editing Perspective

While the functionalities of the procedure editor (cf. figure 7.2) remain the same, the graphical appearance is restructured a lot. It turned out to be impractical to place all elements on the same screen, as it would be too crowded (cf. 4.3.2). Therefore, the procedure editor is organized in several tabs. The first tab "General" contains all general information about the procedure. Its name, general settings, the description, the declaration of procedure parameters as well as pre- and post-conditions. The table for the sequence of events remains the same, but is moved to an own tab and in the lower part the currently selected step can be edited. If the resource control function was implemented, the definition of constraints would be placed in a third tab. The conflict console would be shown as separate part below the actual editor window. On the left-hand side the perspective features the procedure explorer, where the user can select a procedure to be opened in an editor.

7.4.4 SoE Monitor

The main difference regarding the SoE monitor perspective (cf. figure 7.3) is that the different categories of steps (past, current, and upcoming) are no longer separated into different tables but all shown in one (cf. 4.3.2). They are distinguished by their font-style. Past ones are written italic and greyed, while current ones are bold and highlighted with a yellow background, and the upcoming ones are without further style. The table can be set to auto-scroll and the algorithm tries to always reveal the current steps and if possible at least three upcoming and three past ones. Furthermore, similarly to the planning perspective, the top panel features three clocks instead of one.
Figure 7.2: Screenshot of the actual procedure editing perspective. Important GUI elements are highlighted.

Figure 7.3: Screenshot of the actual SoE monitor perspective.
7.5 Deployment

When the implementation is finished, the software product has to be distributed to the users and deployed on their machines. As the RCP and SWT are, other than common Java applications, not completely platform independent, different versions for different target platforms have to be compiled. This process is supported by the IDE and corresponding wizards and explained in detail in [Steppan, 2015, ch. 31], [Vogel, 2016], and [The Eclipse Foundation, 2017]. The wizard not only compiles all sources but also bundles all required external libraries and plug-ins or, if set to do so, the preferred Java Runtime Environment (JRE). Thus, the export guided by the wizard results in a binary folder containing a executable for the corresponding platform and further subfolders containing all the dependencies. To deploy the software on another machine, simply the whole binary folder has to be copied to the system and the executable run. If the export does not contain the JRE, the required one has to be installed on the machine. If the JRE is included, the software is executable completely out of the box. If the JRE is both already installed on the machine and included in the exported binary folder, the one in the binary folder can safely be removed by simply deleting the folder jre in the binary folder to save disk storage or to avoid security issues due to outdated Java installations.

The MPT requires JRE 1.8.0 Build 131, which is included in the binary folder. Although cross-platform deployment is basically supported, it was only tested for Microsoft Windows and thus the MPT is currently only released for this platform. The exported MPT was successfully deployed on different Microsoft Windows machines, running both Windows 7 and Windows 10. The successful deployment tests included to execute the MPT both with the local JRE installation or the 32-bit as well as 64-bit one which were shipped with the MPT.

7.6 Doxygen Documentation

To further prepare future maintenance and extension of the final software product, all source files were commented with javadocs (cf. [Ullenboom, 2014, ch. 22.4]). The source files were then scanned by the Doxygen tool (cf. [van Heesch, 2016]) to assemble a nice technical documentation of the project. It is available both as PDF document and as HTML-version on the attached DVD as well as in the project’s SVN repository (docu folder). For further questions regarding the implementation, please refer to this documentation.

Please note, that the project’s sources contain several classes, which are actually not used in the final version, but could be a good starting point for further developments. Those classes are marked as "CURRENTLY NOT USED" prominently in the class description.
In order to assess the final MPTs functionality, correctness, and completeness measured by the defined objectives and requirements (cf. 3) thorough testing and verification is essential. The first section of this chapter shall outline the applied test methodology and philosophy. In the second section the capabilities of the final software product are checked against the specific requirements. This is mainly done through the review of the design and corresponding source files.

### 8.1 Test Philosophy

The so called test driven development (TDD) is a common philosophy to test software as thoroughly as possible\(^1\). The main idea of the TDD approach is to first define a set of test cases, as complete as possible regarding special cases, at hand of the requirements provided in natural language, before implementation is even started. In the best case this should prevent the developers to miss out critical cases, which is much more likely if the tests are defined after the implementation, when the developers’ minds are already biased by the implementation. Those tests are preferably to be defined in a way which enables automatic tests and assessments if they have passed or not. Depending on how strictly the TDD is followed, in the following steps the implementation is first brought to a level where all sources can be compiled and all tests are executable but fail, and only afterwards each item under test is one by one implemented to pass all tests. If the software has to be restructured or refactored, the refactoring step is only regarded to be finished, when the same state regarding the pass or fail of all tests is achieved as before the refactoring (cf. [Oestereich, 2012, ch. 3.2.14]).

It is obvious, that this approach is basically favourable. However, it also has some drawbacks. The thorough definition of the tests takes a lot of time. As the benefits of a more stable software justify those efforts, this is often acceptable for large projects.

\(^1\)Although it is obvious that "testing shows the presence, not the absence of bugs." [Dijkstra, 1970, p. 7]
But it might be crucial for small and short projects, if the time for thorough design and implementation runs short because a lot of time was spent at the beginning of the project for the definition of the tests. The worst case would be, if time was spent on tests for (optional) features, which then could not be implemented due to the lack of time. Secondly, the (psychological) effect of TDD, to first define the test cases before one has a specific implementation in mind, applies best for teams, where the tests are preferably defined by another person than the one who actually implements the features. This is due the experience, that even when defining the tests and focussing on this with the best intention, most software engineers will have at least a vague idea for the implementation. Thus, the benefits of TDD are severely limited especially for "one-man-projects". Thirdly, TDD with its automatic tests applies best on units of the business logic, where the units under test perform a very specific action (i.e. modification of an object, calculation of a function) provided a specific input and returning it. For units under test, which require user interaction especially elements of graphical user interfaces, are thus very hard to test in automatic manner.

Having this argumentation in mind and the fact that the development of the MPT in the scope of the master project is a one-man-project and contains significant amount of GUI development while the underlying business logic is relatively simple, it is obvious that strictly following TDD is impractical for this project. Nevertheless, to benefit from the underlying idea of the TDD, one sticks to the very basic principles:

- Before the implementation of some software unit is started, one shall think of critical input and special cases.
- Although not defined in a standardized manner, those tests shall be run after the implementation of the software unit.
- The implementation of another unit shall not be started, before the one implemented before does not pass all tests.
- Where applicable, the tests may be run automatically, but also manually, i.e. by manual input and user interaction, or "manual" comparison of return values and expected values.

To round out the testing philosophy a significant amount of time of the project was reserved for beta-tests. For those tests a working version of the software was distributed to the end users and feedback regarding bugs or improvements of features was reported to the developer who could then apply patches before the final release.
8.2 Verification

8.2.1 General Remarks

This section checks the actual capabilities and functions of the final MPT in a systematic manner against the objectives and requirements (cf. 3 and A). The tables list the requirements with increasing abstraction level, i.e. it is started with the most detailed ones, more abstract ones are verified based on the verification of the corresponding children and finally the project statement is verified.

The tables contain the requirements’ ID, the keywords indicating their priorities (cf. 3.4), their status and where applicable an explanation or comment. The status is color coded as follows:

- requirements which are fully met are marked green.
- requirements which are partly met are marked orange.
- compulsory requirements which are not met for a justified reason are marked orange.
- compulsory requirements which are not met without justification are marked red.
- optional requirements which are not met remain unmarked.

8.2.2 Verification of Operational Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Priority</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-O-0010</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-O-0020</td>
<td>may-low</td>
<td>not met</td>
<td></td>
</tr>
<tr>
<td>MPT-O-0030</td>
<td>shall</td>
<td>justified</td>
<td>The RCP application is that stable that no crashes causing data loss were</td>
</tr>
<tr>
<td></td>
<td></td>
<td>not met</td>
<td>experienced during implementation and testing. No special measures were</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>thus be taken for recovery from crashes to prevent data loss.</td>
</tr>
</tbody>
</table>

Table 8.1: Verification matrix for operational requirements.
8.2.3 Verification of Design Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Priority</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-D-0010</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-D-0020</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-D-0030</td>
<td>should - high</td>
<td>not met</td>
<td>The orbit propagator is not implemented.</td>
</tr>
<tr>
<td>MPT-D-0040</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-D-0030</td>
</tr>
<tr>
<td>MPT-D-0050</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-D-0030</td>
</tr>
<tr>
<td>MPT-D-0060</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-D-0030</td>
</tr>
<tr>
<td>MPT-D-0070</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-D-0030</td>
</tr>
<tr>
<td>MPT-D-0080</td>
<td>shall</td>
<td>fully met</td>
<td>Internally Java’s long type (64-bit signed) is used representing the seconds from lift-off.</td>
</tr>
<tr>
<td>MPT-D-0090</td>
<td>shall</td>
<td>fully met</td>
<td>It provides separate timelines for &quot;organisational units&quot; and each procedure.</td>
</tr>
<tr>
<td>MPT-D-0100</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-D-0030</td>
</tr>
<tr>
<td>MPT-D-0110</td>
<td>shall</td>
<td>justified not met</td>
<td>The application runs stable and the user is warned to save changes before exiting. Auto-save functions are thus not regarded necessary.</td>
</tr>
<tr>
<td>MPT-D-0120</td>
<td>shall</td>
<td>fully met</td>
<td>The configuration is saved to a config file, when applied.</td>
</tr>
<tr>
<td>MPT-D-0130</td>
<td>shall</td>
<td>fully met</td>
<td>If a config file is available it is loaded; otherwise hard-coded default values are loaded.</td>
</tr>
</tbody>
</table>

Table 8.2: Verification matrix for design requirements.
8.2.4 Verification of Performance Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Priority</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0010</td>
<td>shall</td>
<td>fully met</td>
<td>see MPT-D-0080</td>
</tr>
<tr>
<td>MPT-P-0020</td>
<td>may - low</td>
<td>fully met</td>
<td>see MPT-D-0080</td>
</tr>
<tr>
<td>MPT-P-0030</td>
<td>shall</td>
<td>fully met</td>
<td>see MPT-D-0080</td>
</tr>
<tr>
<td>MPT-P-0040</td>
<td>shall</td>
<td>fully met</td>
<td>see MPT-D-0080</td>
</tr>
<tr>
<td>MPT-P-0050</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-P-0055</td>
<td>shall</td>
<td>fully met</td>
<td>Steps are always sorted with their timestamp as first and their index in the procedure as second sorting criteria.</td>
</tr>
<tr>
<td>MPT-P-0060</td>
<td>should - high</td>
<td>not met</td>
<td>The resource control remains unimplemented.</td>
</tr>
<tr>
<td>MPT-P-0070</td>
<td>should - high</td>
<td>not met</td>
<td>The orbit propagator remains unimplemented.</td>
</tr>
<tr>
<td>MPT-P-0080</td>
<td>shall</td>
<td>fully met</td>
<td>no rounding was necessary.</td>
</tr>
<tr>
<td>MPT-P-0090</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-P-0070</td>
</tr>
<tr>
<td>MPT-P-0100</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-P-0070</td>
</tr>
<tr>
<td>MPT-P-0110</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-P-0120</td>
<td>does not have to</td>
<td>fully met</td>
<td>Internally, this is allowed but cannot be displayed correctly on the timelines.</td>
</tr>
</tbody>
</table>

Table 8.3: Verification matrix for performance requirements.

8.2.5 Verification of Functional Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Priority</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-F-0010</td>
<td>should - high</td>
<td>not met</td>
<td>The orbit propagator remains unimplemented.</td>
</tr>
<tr>
<td>MPT-F-0020</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0030</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0040</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0050</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0060</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0070</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0080</td>
<td>shall</td>
<td>fully met</td>
<td>via button or menu</td>
</tr>
<tr>
<td>MPT-F-0090</td>
<td>shall</td>
<td>fully met</td>
<td>via context menu or double click</td>
</tr>
<tr>
<td>MPT-F-0100</td>
<td>shall</td>
<td>fully met</td>
<td>via context menu</td>
</tr>
</tbody>
</table>

Table 8.4: Verification matrix for functional requirements.
<table>
<thead>
<tr>
<th>ID</th>
<th>Priority</th>
<th>Status</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-F-0110</td>
<td>shall</td>
<td>fully met</td>
<td>via menu or short-cut</td>
</tr>
<tr>
<td>MPT-F-0120</td>
<td>shall</td>
<td>fully met</td>
<td>The file is automatically loaded at start-up.</td>
</tr>
<tr>
<td>MPT-F-0130</td>
<td>shall</td>
<td>fully met</td>
<td>All classes provide the necessary getter and setter methods.</td>
</tr>
<tr>
<td>MPT-F-0140</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-F-0150</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-F-0160</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0170</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0010</td>
</tr>
<tr>
<td>MPT-F-0180</td>
<td>shall</td>
<td>fully met</td>
<td></td>
</tr>
<tr>
<td>MPT-F-0190</td>
<td>shall</td>
<td>fully met</td>
<td>they can be selected independently; different LT clocks may be defined.</td>
</tr>
<tr>
<td>MPT-F-0200</td>
<td>shall</td>
<td>fully met</td>
<td>in a tabular manner in the SoE monitor perspective.</td>
</tr>
<tr>
<td>MPT-F-0210</td>
<td>shall</td>
<td>fully met</td>
<td>in the preferences dialog</td>
</tr>
<tr>
<td>MPT-F-0220</td>
<td>shall</td>
<td>fully met</td>
<td>in the preferences dialog</td>
</tr>
<tr>
<td>MPT-F-0240</td>
<td>should - medium</td>
<td>not met</td>
<td>The resource control remains unimplemented.</td>
</tr>
<tr>
<td>MPT-F-0250</td>
<td>should - medium</td>
<td>not met</td>
<td>see MPT-F-0240</td>
</tr>
<tr>
<td>MPT-F-0260</td>
<td>should - low</td>
<td>not met</td>
<td>see MPT-F-0240</td>
</tr>
<tr>
<td>MPT-F-0270</td>
<td>should - medium</td>
<td>not met</td>
<td>The procedure validator remains unimplemented.</td>
</tr>
<tr>
<td>MPT-F-0280</td>
<td>should - low</td>
<td>not met</td>
<td>see MPT-F-0240</td>
</tr>
<tr>
<td>MPT-F-0290</td>
<td>may - low</td>
<td>not met</td>
<td>see MPT-F-0240</td>
</tr>
<tr>
<td>MPT-F-0300</td>
<td>may - low</td>
<td>not met</td>
<td>see MPT-F-0240</td>
</tr>
<tr>
<td>MPT-F-0310</td>
<td>should - high</td>
<td>fully met</td>
<td>via menu</td>
</tr>
<tr>
<td>MPT-F-0320</td>
<td>can - low</td>
<td>fully met</td>
<td>provided by RCP plug-ins</td>
</tr>
<tr>
<td>MPT-F-0330</td>
<td>shall</td>
<td>fully met</td>
<td>via button or menu</td>
</tr>
<tr>
<td>MPT-F-0340</td>
<td>shall</td>
<td>fully met</td>
<td>via menu or short-cut</td>
</tr>
<tr>
<td>MPT-F-0350</td>
<td>shall</td>
<td>fully met</td>
<td>via double-click in the procedure explorer</td>
</tr>
<tr>
<td>MPT-F-0360</td>
<td>shall</td>
<td>fully met</td>
<td>in the procedure editor</td>
</tr>
<tr>
<td>MPT-F-0380</td>
<td>shall</td>
<td>fully met</td>
<td>Also other properties are implemented.</td>
</tr>
<tr>
<td>MPT-F-0390</td>
<td>should - low</td>
<td>partly met</td>
<td>The complete mission plan can be exported as comma separated file and thus post-processed e.g. by spreadsheet software for printing.</td>
</tr>
<tr>
<td>MPT-F-0400</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0270</td>
</tr>
<tr>
<td>MPT-F-0410</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0240</td>
</tr>
<tr>
<td>MPT-F-0420</td>
<td>should - high</td>
<td>not met</td>
<td>see MPT-F-0240</td>
</tr>
<tr>
<td>MPT-F-0430</td>
<td>shall</td>
<td>fully met</td>
<td>The only part external from the MPT is the telecommand database. An interface towards it is implemented.</td>
</tr>
</tbody>
</table>

Table 8.5: Verification matrix for functional requirements (continued).
8.2.6 Verification of the Project Objectives & Project Statements

From the verification of the requirements one can directly deduct, that the primary objectives MPT-1PO-0020 to MPT-1PO-0050, i.e. the design and implementation of the procedure editor, the scheduling timeline, the command list generator, and the corresponding intuitive GUI, were achieved. MPT-1PO-0060, the development and design of concepts for the mission planning for highly autonomous satellites, did not pose any requirements onto the development of the MPT itself, but was handled separately as a theoretical component of this thesis (cf. 5). Please note, that the priority of MPT-1PO-0010, the design and implementation of the orbit propagator, was reduced to a secondary objective compared to the first version of the project definition.

While all primary objectives were achieved, this is not the case for any of the secondary objectives. However, during the analysis and early design stages, all were regarded and the MPT kept open for extensions in general. Thus, valuable preparations for those components, which are good starting points for further developments, were achieved in the scope of this thesis project.

As the secondary objectives are to no extent essential for the operations of SONATE and all of the primary objectives were achieved, the project as a whole (cf. MPT-PS-0010) is to be regarded as a full success.
This page is intentionally left blank.
Although the project is regarded as full success, various starting points for further developments towards a powerful generic MPT for small satellite missions can be identified. They are grouped into four fields and shortly outlined in the following.

The first field of further development covers all possible improvements which are not related to features of the MPT but the software quality in general. The application runs perfectly stable, i.e. causing no loss of data due to crashes, but the testing revealed few nasty bugs, which only seldom occur and therefore are hard to reproduce and fix, which could not be fixed in the scope of this project. Furthermore, the modularity could further be increased. For instance, the whole application is currently one RCP plug-in. To increase the modularity, the telecommand component could be outsourced to a separate headless plug-in, thus increasing the portability of the MPT. This restructuring was omitted because it was not required and would have added significant effort for development and testing.

The second field is the implementation of further features. This includes especially all features of the secondary objectives, the orbit propagator, resource control, procedure validator, or the concepts for the mission planning for autonomous satellites. It could of course extend much beyond that to even more advanced features like automatic conflict resolution, automatic planning, multi-user support and so on. While most of it is desirable from the point of view of mission planning, it is important to keep the general complexity for the usage of the MPT low. Otherwise, one would miss the vision and the unique selling point of a generic and powerful planning tool which is yet applicable also for small satellites, and the MPT would thus become "just another MPT" standing side-by-side with the gigantic mission planning systems of the large agencies, but probably never reaching their capabilities.

The third field regards the graphical user interface. Already the beta-tests showed that, while the interface is fully functional, there are plenty of possible improvements which would render the interface even more user friendly and ergonomic. Those possible improvements range from really easy ones, like providing icons for all important controls and menu items, or more and more detailed help information or tooltips, to more tricky ones like persisting the window configuration between separate sessions,
to really advanced ones like drag and drop capabilities of the scheduling timelines.

The fourth and last field for further work takes care of the emerging sector of autonomous satellites. This thesis approached this topic only in a very theoretical way. Besides the implementation of the concepts as part of the MPT, which would fall into the second field of further work, more research, perhaps empirical, compared to the implementation yet theoretical, could be conducted regarding the question of the implications for mission planning for autonomous satellites. For instance, the presented concepts could be elaborated and designed in detail, or their actual usability and practicability could be further examined. As the literature review suggested that the topic is not yet covered by the large agencies’ mission planning departments, broadly assessing the characteristics and implications for mission planning of recent projects in the field of high autonomy for space missions would be a very basic but reasonable and important aspect as well.

All of those various starting points for further work on the MPT have in common that they can basically be arbitrarily scaled. Thus, the further work could be conducted as a seminar project, covering only a very detailed aspect, a bachelor project, which would for instance implement one of the secondary objectives, another master thesis, which could conduct the theoretical research on the aspect of autonomy, as well as a doctoral thesis, which would work on all aspects and the MPT as a whole. To summarize, the topic of the development of a generic and powerful MPT applicable for small satellites is not extend "grazed", there are plenty of possibilities to conduct further research in this field and it is strongly encouraged to do so.
In the scope of this thesis work, a generic and flexible mission planning tool for SONATE was developed. It is based on the Eclipse Rich Client Platform and features a procedure editor, a planning timeline and a sequence of events monitor, thus supporting the main use cases for the systems engineer, planning engineer, and operator. Although none of the secondary objectives could be achieved, they were regarded during the early analysis and design steps. So the final software product remains open for extension of advanced features like the resource control, procedure validator, or perhaps even automatic conflict resolution. The MPT, which was successfully deployed on several Windows machines, can be adapted for other missions, e.g. the envisioned 27U-nanosat follow-up mission of SONATE or other small satellite missions, with minimal development effort. This is thanks to its generic approach, i.e. that the mission model is defined separately from the core functionalities of the MPT. Taking also into account the various starting points for further developments, this thesis work not only supports the SONATE mission, but the development of a generic timeline-based MPT applicable and affordable for small satellite missions in general was kicked off.

Besides the development of the MPT itself, important questions regarding the implications of highly autonomous satellites for the mission planning, were discussed for the first time in the scope of this thesis. To face the challenge to combine the classical mission planning approach with the emerging highly autonomous systems in the space sector, three concepts, namely Safe Planning, Sigma Resource Propagation, and Direct Telemetry Feedback, were proposed and theoretically discussed.

In total this project both contributes to the scientific community of mission planning and profoundly supports the future mission operations of SONATE. Personally, I am glad that I could help SONATE’s future operational team not to fail its planning task and could thus ultimately contribute to the expected overall success of the mission.
This page is intentionally left blank.
## A.1 Functional Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-F-0010</td>
<td>The MPT should provide a method to input orbit parameters.</td>
<td>MPT-1PO-0010</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>As the design and implementation of the orbit and ground contact propagator is only an optional secondary objective, all requirements regarding the resource control function become optional.</td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td><strong>Value:</strong></td>
<td></td>
<td><strong>Priority:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>MPT-F-0020</td>
<td>The MPT should calculate the current position of the satellite using the orbit parameters.</td>
<td>MPT-1PO-0010</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>See comment of MPT-F-0010.</td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td><strong>Value:</strong></td>
<td></td>
<td><strong>Priority:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>MPT-F-0030</td>
<td>The MPT should propagate the position of the satellite.</td>
<td>MPT-1PO-0010</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>See comment of MPT-F-0010.</td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td><strong>Value:</strong></td>
<td></td>
<td><strong>Priority:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>MPT-F-0040</td>
<td>The MPT should calculate whether the satellite is currently on the daylight side or in eclipse.</td>
<td>MPT-1PO-0010</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>See comment of MPT-F-0010.</td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td><strong>Value:</strong></td>
<td></td>
<td><strong>Priority:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>-----------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>MPT-F-0050</td>
<td>The MPT should predict the phases of daylight and eclipse phases.</td>
<td>MPT-1PO-0010</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-F-0010.</td>
<td>Value: Priority:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>MPT-F-0060</td>
<td>The MPT should provide a method to model several ground stations and their field of view.</td>
<td>MPT-1PO-0010</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-F-0010.</td>
<td>Value: Priority:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>MPT-F-0070</td>
<td>The MPT should predict the visibility of the satellite from the ground stations for ground contacts.</td>
<td>MPT-1PO-0010</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-F-0010.</td>
<td>Value: Priority:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>MPT-F-0080</td>
<td>The MPT shall enable the user to insert new timeline entries to the mission plan.</td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>MPT-F-0090</td>
<td>The MPT shall enable the user to reschedule existing timeline entries in the mission plan.</td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>MPT-F-0100</td>
<td>The MPT shall enable the user to delete existing timeline entries from the mission plan.</td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>MPT-F-0110</td>
<td>The MPT shall enable the user to save the mission plan to a file.</td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>MPT-F-0120</td>
<td>The MPT shall enable the user to load an existing mission plan from a file.</td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0130</td>
<td>The MPT shall provide interfaces to enable access to the mission plan for resource control functions.</td>
<td>MPT-1PO-0030 MPT-2PO-0010</td>
</tr>
<tr>
<td>Comment:</td>
<td>Although the design and implementation of resource control functions are only a secondary project objective, it is strictly required (“shall”) to provide suitable interfaces not to obstruct future developments even if MPT-2PO-0010 is not fulfilled.</td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0140</td>
<td>The MPT shall generate a telecommand sequence of the mission plan in a format which is compatible to the telecommand client.</td>
<td>MPT-1PO-004</td>
</tr>
<tr>
<td>Comment:</td>
<td>For the definition of compatible input formats for the telecommand tool refer to [Beylin, 2014].</td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0150</td>
<td>The MPT shall generate the telecommand sequence of the mission plan for a user specified time interval.</td>
<td>MPT-1PO-0040</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-1PO-0010</td>
<td>Value: Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0160</td>
<td>The GUI of the MPT should graphically visualize daylight and eclipse phases on a timeline.</td>
<td>MPT-1PO-0050 MPT-1PO-0010</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-1PO-0010</td>
<td>Value: Priority: high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0170</td>
<td>The GUI of the MPT should graphically visualize contact periods to the ground stations on timelines.</td>
<td>MPT-1PO-0050 MPT-1PO-0010</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-1PO-0010</td>
<td>Value: Priority: high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>MPT-F-0180</td>
<td>The GUI of the MPT shall graphically visualize the planned timeline entries</td>
<td>MPT-1PO-0050</td>
</tr>
<tr>
<td></td>
<td>of the space segment on timelines.</td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td></td>
<td><strong>Comment:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-F-0190</td>
<td>The GUI of the MPT shall graphically visualize different time axes.</td>
<td>MPT-1PO-0050</td>
</tr>
<tr>
<td></td>
<td><strong>Comment:</strong></td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td></td>
<td>Value: LT, UTC, MET</td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-F-0200</td>
<td>The GUI of the MPT shall textually visualize the mission plan.</td>
<td>MPT-1PO-0050</td>
</tr>
<tr>
<td></td>
<td><strong>Comment:</strong></td>
<td>MPT-1PO-0030</td>
</tr>
<tr>
<td></td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-F-0210</td>
<td>The GUI of the MPT shall visualize the applied settings of the MPT.</td>
<td>MPT-1PO-0050</td>
</tr>
<tr>
<td></td>
<td><strong>Comment:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-F-0220</td>
<td>The GUI of the MPT shall enable the user to change all relevant settings of</td>
<td>MPT-1PO-0050</td>
</tr>
<tr>
<td></td>
<td>the MPT.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Comment:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-F-0230</td>
<td><em>removed</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Comment:</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Value:</td>
<td>Priority:</td>
</tr>
</tbody>
</table>

**LT** = Local Time; **UTC** = Coordinated Universal Time; **MET** = Mission Elapsed Time
### A.1. Functional Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-F-0240</td>
<td>The GUI of the MPT should graphically visualize the state and its time-dependent course of resources on timelines.</td>
<td>MPT-1PO-0050 MPT-2PO-0010</td>
</tr>
<tr>
<td><strong>Comment</strong>:</td>
<td>As the design and implementation of the resource control function is only an optional secondary objective, all requirements regarding the resource control function become optional.</td>
<td><strong>Value:</strong> medium <strong>Priority:</strong></td>
</tr>
<tr>
<td>MPT-F-0250</td>
<td>The GUI of the MPT should textually visualize conflicts detected by the resource control.</td>
<td>MPT-1PO-0050 MPT-2PO-0010</td>
</tr>
<tr>
<td><strong>Comment</strong>:</td>
<td>See comment of MPT-F-0240.</td>
<td><strong>Value:</strong> medium <strong>Priority:</strong></td>
</tr>
<tr>
<td>MPT-F-0260</td>
<td>The GUI of the MPT can graphically visualize conflicts detected by the resource control.</td>
<td>MPT-1PO-0050 MPT-2PO-0010</td>
</tr>
<tr>
<td><strong>Comment</strong>:</td>
<td>See comment of MPT-F-0240.</td>
<td><strong>Value:</strong> low <strong>Priority:</strong></td>
</tr>
<tr>
<td>MPT-F-0270</td>
<td>The GUI of the MPT should textually visualize conflicts detected by the procedure validator.</td>
<td>MPT-1PO-0050 MPT-2PO-0020</td>
</tr>
<tr>
<td><strong>Comment</strong>:</td>
<td>As the design and implementation of the procedure validation function is only an optional secondary objective, all requirements regarding the procedure validation function become optional.</td>
<td><strong>Value:</strong> medium <strong>Priority:</strong></td>
</tr>
<tr>
<td>MPT-F-0280</td>
<td>The GUI of the MPT should textually visualize conflicts detected by the procedure validator.</td>
<td>MPT-1PO-0050 MPT-2PO-0020</td>
</tr>
<tr>
<td><strong>Comment</strong>:</td>
<td>See comment of MPT-F-0270.</td>
<td><strong>Value:</strong> low <strong>Priority:</strong></td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-------------------------</td>
</tr>
</tbody>
</table>
| MPT-F-0290  | The MPT may provide a GUI for the definition of rules for the resource control. | MPT-1PO-0050
                                |                                                                      | MPT-2PO-0010            |
| **Comment:**| As the design and implementation of the resource control function is only an optional secondary objective, all requirements regarding the resource control function become optional. The priority is low because the definition of rules is no routine task but done once per mission and thus higher effort due to the lack of a graphical interface is justified. | **Value:**
                                |                                                                      | **Priority:** low       |
| MPT-F-0300  | The MPT may provide a GUI for the definition of rules for the procedure validator. | MPT-1PO-0050
                                |                                                                      | MPT-2PO-0020            |
| **Comment:**| As the design and implementation of the procedure validator function is only an optional secondary objective, all requirements regarding the procedure validator function become optional. The priority is low because the definition of rules is no routine task but done once per mission and thus higher effort due to the lack of a graphical interface is justified. | **Value:**
                                |                                                                      | **Priority:** low       |
| MPT-F-0310  | All controls and inputs of the GUI of the MPT should be lockable.         | MPT-1PO-0050
                                |                                                                      | M-0120                  |
| **Comment:**| During various use cases, i.e. when the MPT is used to supervise already planned timeline entries, no user input will be needed. Contrary, unintended user input might be a risk to the mission during those use cases. To prevent unintended user input, all controls and input should be lockable. | **Value:**
<pre><code>                            |                                                                      | **Priority:** high      |
</code></pre>
<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-F-0320</td>
<td>The window configuration of the GUI of the MPT can be flexible.</td>
<td>MPT-1PO-0050 M-0120</td>
</tr>
<tr>
<td>Comment: In different use cases, different information is more important than other data. To focus the operator’s attention to the data which is relevant for the task he is conducting, it would be an asset if he could resize and place windows respectively show and hide information to his personal needs and preferences.</td>
<td>Value:</td>
<td>Priority: low</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0330</td>
<td>The MPT shall enable the user to create new procedures.</td>
<td>MPT-1PO-0020</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0340</td>
<td>The MPT shall enable the user to save procedures to a file.</td>
<td>MPT-1PO-0020</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0350</td>
<td>The MPT shall enable the user to load existing procedures from a file.</td>
<td>MPT-1PO-0020</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0360</td>
<td>The MPT shall enable the user to change existing procedures.</td>
<td>MPT-1PO-0020</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0370</td>
<td>removed</td>
<td></td>
</tr>
<tr>
<td>Comment: Duplicate</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-F-0380</td>
<td>A procedure shall consist of: a set of prerequisites with or without timing condition; an ordered and timed set of steps; a set of pass and fail criteria; a set of comments.</td>
<td>MPT-1PO-0020</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>MPT-F-0390</td>
<td>The MPT should provide printable output of procedures and their scheduled instances.</td>
<td>MPT-1PO-0020 M-0120</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>While supervising the execution of timeline entries, it might be handy for the operator or even required for reasons of legal documentation to have a printed version of the procedures (respectively the timeline entry under execution) in the style of a checklist.</td>
<td><strong>Value:</strong> low</td>
</tr>
<tr>
<td>MPT-F-0400</td>
<td>The MPT should validate user generated procedures against predefined rules.</td>
<td>MPT-1PO-0020 M-0120</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>See comment of MPT-F-0270.</td>
<td><strong>Value:</strong> high</td>
</tr>
<tr>
<td>MPT-F-0410</td>
<td>The MPT should propagate the state of resources according to predefined rules.</td>
<td>MPT-1PO-0010 M-0120</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>See comment of MPT-F-0240.</td>
<td><strong>Value:</strong> high</td>
</tr>
<tr>
<td>MPT-F-0420</td>
<td>The MPT should check the propagated future state of resources against predefined rules.</td>
<td>MPT-1PO-0010 M-0120</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>See comment of MPT-F-0240.</td>
<td><strong>Value:</strong> high</td>
</tr>
<tr>
<td>MPT-F-0430</td>
<td>The MPT shall implement interfaces to the mission information base.</td>
<td>MPT-1PO-0010 MPT-1PO-0020 MPT-1PO-0040 MPT-2PO-0010 MPT-2PO-0020</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>The mission information base is the model of the mission containing all data which is relevant for the mission operation in general. It consists of various sub-bases, like the telecommand database, telemetry database, orbit parameters, ground station models, rules for the resource control, etc. Its precise content and structure is not yet specified.</td>
<td><strong>Value:</strong></td>
</tr>
</tbody>
</table>
## A.2 Performance Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0010</td>
<td>The scheduling timeline shall extend to at least [ ]</td>
<td>MPT-C-0050</td>
</tr>
<tr>
<td>Comment:</td>
<td>The mission is designed for 1 year in orbit. Including pre-launch activities which might be also planned with the MPT and a margin, a minimum value of 1.5 year is justified.</td>
<td>Value: 1.5 year</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0020</td>
<td>The scheduling timeline may extend until the expected reentry of the satellite.</td>
<td>MPT-C-0050 M-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td>It is quite common that well-designed pico satellite missions outperform their expected lifetime. Therefore, the MPT may also support an extended mission.</td>
<td>Value:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0030</td>
<td>The time resolution of the scheduling timeline shall be [ ]</td>
<td>OBDH-0540</td>
</tr>
<tr>
<td>Comment:</td>
<td>OBDH-0540 requires a time synchronization offset between space and ground segment of ±2 s. The time resolution of the MPT shall be in the same order of magnitude. The value of 1 s was chosen due to simplicity.</td>
<td>Value: 1 s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0040</td>
<td>The time resolution of different steps of a procedure shall be [ ]</td>
<td>OBDH-0540</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-P-0030.</td>
<td>Value: 1 s</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>MPT-P-0050</td>
<td>The MTP shall support steps to be scheduled in a procedure with the same time tag.</td>
<td>MPT-P-0040, M-0120</td>
</tr>
</tbody>
</table>

**Comment:**
As the time resolution of steps is only 1 sec but the execution of some may take less time, it may be allowed to have multiple steps with the same time tag.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0055</td>
<td>If two steps are scheduled in a procedure with the same time tag, the MPT shall guarantee that their order remains stable throughout the whole processing.</td>
<td>MPT-P-0040, MPT-P-0050, M-0120</td>
</tr>
</tbody>
</table>

**Comment:**
While short steps may have the same time tag, their order of execution may be semantically relevant and decisive for proper operations. Therefore, the MPT shall guarantee that step A which is scheduled with the same time tag but before step B, will also be executed first.

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0060</td>
<td>The time resolution of the resource control should be [ ]</td>
<td>OBDH-0540</td>
</tr>
</tbody>
</table>

**Comment:**
Applies only if the resource control function is implemented. See also comment of MPT-P-0030

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0070</td>
<td>The time resolution of the orbit and ground contact propagator should be [ ]</td>
<td>OBDH-0540</td>
</tr>
</tbody>
</table>

**Comment:**
Only applies, if the orbit propagator is implemented. Internally a higher resolution might be necessary to achieve reliable prediction, while the results shall be delivered with the stated resolution. See also comment of MPT-P-0030.
<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-P-0080</td>
<td>Where rounding of results is necessary to match the required time resolution, the MPT shall round conservatively towards the worst case.</td>
<td>M-0120</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>That means that the availability of resources shall be underestimated, while the duration of negative effects (like eclipse times) shall be overestimated.</td>
<td>Value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-P-0090</td>
<td>The prediction of ground contacts (AOS, LOS) should not differ more than values compared to the actual events.</td>
<td>MPT-F-0070</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>Only applies, if the orbit propagator is implemented. This might be tested using professional orbit and ground contact propagators (like STK) as reference or by experiments with satellites which are already in orbit.</td>
<td>Value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-P-0100</td>
<td>The duration of daylight phases should be modeled conservatively.                                                                PLEMENTED.</td>
<td>MPT-F-0050</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>Only applies, if the orbit propagator is implemented. It shall not be predicted to be longer than compared to the prediction of professional orbit propagators (like STK).</td>
<td>Value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Priority:</td>
</tr>
<tr>
<td>MPT-P-0110</td>
<td>The MPT shall allow concurrent timeline entries between different payloads and relevant subsystems of the satellite bus.</td>
<td>N-0010</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>While an timeline entry is scheduled for payload A, it shall be possible to schedule another timeline entry for payload B for the same time.</td>
<td>Value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Priority:</td>
</tr>
</tbody>
</table>
### A.3 Design Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-D-0010</td>
<td>The MPT shall be implemented in Java.</td>
<td>MPT-C-0020</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MPT-C-0030</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>When a timeline entry is scheduled for payload A,</td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td></td>
<td>the MPT does not have to support the scheduling of another timeline entry</td>
<td><strong>Priority:</strong></td>
</tr>
<tr>
<td></td>
<td>for payload A at the same (or overlapping) time.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-D-0020</td>
<td>The MPT shall use XML-formats to save user input (mission plan, procedures, settings).</td>
<td>MPT-C-0080</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td></td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Priority:</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-D-0030</td>
<td>The MPT should accept the orbit parameters from TLEs.</td>
<td>MPT-F-0010</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>Only applies, if the orbit propagator is implemented.</td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Priority:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-D-0040</td>
<td>The MPT should allow direct input of the orbit parameters.</td>
<td>MPT-F-0010</td>
</tr>
<tr>
<td><strong>Comment:</strong></td>
<td>See comment of MPT-D-0030.</td>
<td><strong>Value:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Priority:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>MPT-D-0050</td>
<td>The MPT should implement the SGP4 model for orbit propagation.</td>
<td>MPT-F-0020, MPT-F-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-D-0030.</td>
<td>Value: high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-D-0060</td>
<td>The MPT should be able to load TLEs from a defined online source.</td>
<td>MPT-D-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-D-0030.</td>
<td>Value: high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-D-0070</td>
<td>The MPT should be able to load TLEs from a local path.</td>
<td>MPT-D-0030</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-D-0030.</td>
<td>Value: high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-D-0080</td>
<td>The MPT shall use appropriate datatypes to provide unique timestamps with a resolution of [ ] for a period of [ ].</td>
<td>MPT-P-0010, MPT-P-0030, MPT-P-0040, MPT-P-0060, MPT-P-0070</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
<td>Value: 1s; 1.5year</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-D-0090</td>
<td>The MPT shall provide a separate scheduling timeline for each payload and relevant subsystem of the satellite bus.</td>
<td>MPT-P-0110, MPT-P-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
<td>Value:</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>MPT-D-0100</td>
<td>The model of a ground station should consist of the following information: geographical coordinates; min, max elevation; min, max azimuth; supports RX and/or TX; supports UHF and/or S-Band</td>
<td>MPT-F-0060</td>
</tr>
<tr>
<td>Comment:</td>
<td>See comment of MPT-D-0030.</td>
<td>Value: high</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Source</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>MPT-D-0110</td>
<td>The MPT shall regularly auto-save conducted changes to temporary files.</td>
<td>MPT-C-0060</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-D-0120</td>
<td>The MPT shall automatically save all relevant settings to configuration files.</td>
<td>MPT-C-0060</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-D-0130</td>
<td>The MPT shall automatically load all relevant settings from configuration files.</td>
<td>MPT-C-0060</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
</tbody>
</table>

### A.4 Operational Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-O-0010</td>
<td>The MPT shall warn the operator for critical input with security alerts.</td>
<td>M-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-O-0020</td>
<td>The MPT may provide functions to undo unintended user input.</td>
<td>M-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority: low</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPT-O-0030</td>
<td>The MPT shall recover from crashes without data loss.</td>
<td>MPT-C-0060</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M-0120</td>
</tr>
<tr>
<td>Comment:</td>
<td>Value:</td>
<td>Priority:</td>
</tr>
</tbody>
</table>
APPENDIX B

User Handbook

While the better part of the thesis itself covers the development of the MPT, this appendix shall serve as an introductory handbook for users. The first section gives a brief and broad overview over the MPT. Secondly, the installation process and first system start-up are explained. The main part of the handbook is then structured along the perspectives and the use-cases implemented by them.

### B.1 Overview

The MPT consists of three main perspectives. The Planning Perspective, the Procedure Editing Perspective, and the SoE Monitor Perspective. Independently from the perspective a main menu can be accessed. Menu entries handling changes of the mission plan are located under File and ones regarding the MPT itself under Window.

The perspectives can be switched with the controls right beneath the main menu (cf. figure B.1). The MPT always launches into the planning perspective and always restores the default layout of the GUI (open parts, size of the parts, content of the parts) at every system start. However, when perspectives are switched while the MPT is running, the GUI layout of each perspective is restored when switching back.

![Figure B.1: Main menu and perspective switch.](image-url)
B.2 Installation & First Execution

B.2.1 Installation

The MPT is shipped as zip-archive and includes all required dependencies and even a Java Runtime Environment. Currently Windows 32-bit and 64-bit is supported. Extract the corresponding archive to a local folder and make sure that you have permissions to read, write and execute on this location.

If a Java Runtime Environment (jre 1.8.0 build 131 or greater) is already installed on the machine, you may delete the folder jre (cf. figure B.2) and all its contents within the installation of the MPT. This saves data storage (about 140 MB) and reduces security risks due to outdated Java installations. If you are not sure whether a correct version of the JRE is already installed, simply keep the folder.

Figure B.2: Binary directory of the MPT. The executable and JRE folder are highlighted.

B.2.2 First Execution

Launch MissionPlanningTool.exe, which is in the root directory of the installation. At the first execution, several error prompts will open. "Don’t panic!" [Adams, 1979] They indicate that several important files are missing. This behaviour is nominal, as the MPT itself generates the files when the mission plan is changed and saved or new preferences are applied. This can of course not be done before the first execution. If the prompts occur although you have already used the MPT, regard them as errors! Now you can use the MPT as you like. Probably you wish to set some preferences as the first step (cf. B.3).

B.3 Preferences

The preferences dialog can be opened via the main menu Window → Preferences and has three main tabs Planning Tool, Mission, and Clocks. Changed settings can be applied via the Apply-Button (dialog remains opened), or OK (dialog window closes) or can be discarded with Cancel. Some of the settings apply directly, while others
require a restart of the MPT. If you are not sure if your changes apply directly, conduct a restart of the MPT.

Figure B.3: Overview of the Preferences Dialog. Tabs and controls are highlighted. The Planning Tool tab is opened.

B.3.1 Planning Tool

This tab (cf. figure B.3) contains all settings which are related to the MPT itself.

TC Database

The location of the TC database (in the form IP:port, e.g. 127.0.0.1:335) can be specified here as well as the corresponding user of this database. The MPT tries to fetch the database once at the system start-up and then holds a local copy of it. If the connection at start-up failed or one wants to refresh the local copy (i.e. when the TC database changed since the start-up of the MPT), the TC database can be fetched (again) by clicking Fetch TC DB. The label TC DB status indicates if the TC database was successfully loaded, and if so when it was loaded.

Root Path

The MPT stores several files of user data (the mission plan itself, the procedures, the configured clocks). The path where those are located can be specified here. Please specify an absolute path; to browse through your file system click the button labelled
"...". The default is <installation>/workspace/mpt where <installation> is the path to your installation of the MPT. It is recommended to separate the program data from the user data, i.e. to set the root path to some user-directory. It can be beneficial to specify a location which can be accessed from several machines (e.g. a network drive) in case the user data shall be accessed from different machines, e.g. primary planning workstation and backup workstation. Please note, that the MPT does not support multi-user-access, i.e. the files must not be accessed from different installations of the MPT at the same time, but in a otherwise coordinated manner!

If the root path is changed, make sure to manually move the contents of the old root path to the new one. Otherwise, the data will not be available in the MPT anymore. To finally apply a new root path please restart the MPT after confirming the preferences dialog.

**GUI**

The minimal width of timeline entries in the planning timelines (cf. B.5.2) can be specified here. This is due to the fact, that depending on the monitor (size and resolution) and the zoom level of the timeline (cf. B.5.1) it may happen that short timeline entries are to small to be drawn and would thus disappear. This can be prevented by setting a sufficient large minimal width here. Please note, that on the other hand too large values will distort the display of longer timeline entries and at higher zoom levels and thus obstructs the inspection of the mission plan via the timelines. So thoroughly balance this setting according to your monitor settings and the shortest possible timeline entry.

**Fraction for "now-line"**

When the timelines are set to auto-scroll (cf. B.5.1), they automatically reveal a certain range around the current time, which is indicated by a red line. The fraction where the current time shall be placed in the plot (measured from the left), i.e. how much of the past and the future is shown (cf. figure B.4), can be set here.

**App Title**

The application’s title can arbitrarily be set here (cf. figure B.5). This might be beneficial when several instances of the MPT are installed for multi-mission operations. Restart the MPT to apply changes of this setting.
B.3. Preferences

(a) 20%

(b) 50%

Figure B.4: Different settings for the fraction for the "now-line".

Figure B.5: Preferences Dialog. Setting the App Title.
B.3.2 Mission

Mission relevant preferences can be set here (cf. figure B.6).

Organisational Units

Here the organisational units of the mission can be defined. The MPT provides a separate timeline for each organisational unit (cf. B.5.1) and procedures can be marked to occupy a organisational unit (cf. B.4.2). Use the buttons Add and Delete to create new ones or delete existing ones and click and edit directly in the table to alter an existing one. Organisational units might be subsystems or payloads in the classical sense, but are not limited to this and could also represent a combination of subsystems or even systems of the ground segment.

Routine Periods

Some time periods are of importance for the periodic planning process. For instance, the planning might commonly be conducted two weeks in advance, hence the planning horizon (cf. B.5.5) shall reach at least two weeks into the future. Similarly, the upload of the plan to the satellite might be performed every two days, thus the SoE export (cf. B.5.4) shall be up-to-date until two days into the future. If those periods are violated, the MPT warns the planning engineer (cf. B.5.6). The exact periods can be specified here.

Figure B.6: Preferences Dialog. Mission tab.
B.3.3 Clocks

The last tab of the preferences dialog covers the clocks of the MPT (cf. figure B.7).

Time Zero

Here the UTC time of the Time Zero singleton, i.e. the time which is associated with the mission elapsed time 0 or differently spoken, the launch date, can be set. Please note, that the MPT internally calculates all times as seconds relative to this UTC date. Changing the date has various side effects and applies properly only after a restart of the MPT. Thus, it shall only be set once, to synchronize the clocks and preferably with an empty mission plan. Unintended changes are protected by the enable checkbox and a warning prompt.

Local Time Clocks

Besides the UTC and MET clocks, the MPT can display arbitrary local clocks in the timelines. They can be defined here (add and delete by buttons, change directly in the table). The first element of the table is regarded as the main local clock and shown in the top panels of the planning perspective and SoE monitor perspective (cf. B.5.1 B.6).
B.4 Procedure Editing Perspective

The procedure editing perspective (cf. figure B.8) consists of the procedure explorer on the left hand side, and arbitrary many editor windows on the right hand side. Each editor window consists of the two tabs General and Sequence of Events.

B.4.1 Create New Procedure

A new procedure can be created via the main menu File → New Procedure. This command switches to the procedure editing perspective (if not already there) and opens a new editor window for a new procedure.

B.4.2 Edit Procedure

The new procedure can then be directly edited in the editor window. Existing procedures can be selected by traversing through the tree of the procedure explorer and opened in a editor window by double-click. If the procedure is already opened in an editor no new editor window is opened but the open one is brought to the foreground.

Figure B.8: Overview of the Procedure Editing Perspective. The General tab of the editor is opened.
B.4. Procedure Editing Perspective

when double-clicking a procedure in the procedure explorer.

General
In the General tab (cf. figure B.8), basic information about the procedure is shown and can be edited.

Name and Path displays the path of the procedure within the procedure repository and its name. As this information is the unique identifier of a procedure, it may only be set once, when saving a new procedure. Therefore, the corresponding field is read only.

Min & Max Duration A minimum and a maximum duration can be specified. When a timeline entry is scheduled on the basis of this procedure it must comply to those durations. The minimum duration must be at least 1s, the maximum must be at least as long as the minimum. A value of 0 for the maximum duration represents positive infinity.

Immediate Tick this option, if the procedure shall only be conducted as immediate procedure (i.e. with immediate instead of time-tagged telecommands only). Among others, this could be the case for procedures with loops or branches, where the operator dynamically has to decide which one shall be taken.

Autonomous Tick this option, if the procedure contains phases where the highly autonomous systems take over control of the satellite. Currently, this field has no further functionality, but shall be modelled consequently to facilitate further implementations of the concepts presented in chapter 5.

Occupied Organisational Units Select all organisational units (they are defined in the preferences; cf. B.3.2) which are occupied by this procedure. When a timeline entry of this procedure is scheduled it will appear in the timelines of all organisational units selected here (cf. B.5.2).

Description, Preconditions, Pass Criteria & Fail Criteria An informative description in natural language about the aims and methods shall be defined for each procedure. Furthermore, preconditions, pass criteria, and fail criteria may be defined in natural language as well. They are not processed automatically in any manner but only serve the planning engineer to assemble a valid mission plan and to assess quickly whether a procedure was conducted successfully or failed.

Parameters Procedure Parameters may be defined here. Procedure parameters are placeholders, which have to be defined in the General tab and can then be used at several fields in the sequence of events of a procedure. When a timeline entry
based on the procedure is scheduled, a value has to be assigned to each parameter and is inserted in the sequence of events in place of the placeholders. The name of a procedure parameter must start with @, consist of at least one other character and must be unique in the scope of the procedure. A parameter can be either of the type integer, float or string. Where applicable a physical unit can be assigned and a meaningful comment shall be provided for each parameter, which also shall state possible ranges/values of the parameter where not obvious.

Sequence of Events

In the Sequence of Events tab (cf. figure B.9) of the procedure editor, the steps which shall be conducted during the procedure are displayed and can be edited.

The top part shows the sequence of events in a tabular manner. In the middle part, the order of the steps can be modified by moving the currently selected step up or down. The marked step can be deleted or a new step inserted above or below the marked one. In the lower part the currently selected step can be edited.

Start The start of the step is specified here. It can either be referenced to the start of the procedure (S+) or the end of the procedure (E-). If S+ is selected, the integer of the following field is interpreted as "seconds after the start of the procedure". If E- is selected, it is interpreted as "seconds before the end of the procedure". Instead of an integer literal also the placeholder of a procedure parameter can be inserted into this field.

Duration An integer duration in seconds can be assigned to a step. Instead of an integer literal also the placeholder of a procedure parameter can be inserted into this field.
**Description** A meaningful description of the step in natural language shall be provided for each step. It may also contain further information or commands to the operator.

**TC & TC Params** Each step may have up to one attached telecommand. It is specified here. If a local copy of the telecommand database is available, this field provides suggestions for auto completion when hitting Ctrl+Space. In the field TC Params, the parameters for the telecommand, as defined in the telecommand database, shall be provided. Several TC parameters are comma-separated. If a local copy of the telecommand database is available and the field TC contains a valid command word, hints for the TC parameters are shown below the field. Instead of literals for the TC parameters also the placeholders of procedure parameters can be inserted.

**Result** The expected result of the step, e.g. direct telemetry feedback, can be defined in natural language in this field.

**General Remarks on the Sequence of Events** As the procedure validator is not implemented, the system engineer who is defining the sequence of events is solely responsible for the correctness of the sequence of events! This especially includes, that

1. the order of the steps corresponds their timing; especially if mixed references (S+ and E-) are used or if the start or duration of some steps is parametrized.
2. only valid telecommand words are used.
3. the provided list of TC parameters matches the definition of the TC database.
4. only parameter placeholders are used which are defined in the General tab.
5. the type of a defined procedure parameter matches its usage.

Not regarding those requirements may lead to exceptions and malfunction of the MPT, or even worse lead to a syntactically correct, but semantically faulty mission plan which is uploaded to the satellite and poses a significant threat onto the mission!

**B.4.3 Save Procedure**

If a procedure was changed, this is indicated by a * in the editor’s tab (cf. figure B.10). When the editor window is selected it can be saved via the main menu File → Save or Ctrl+S. In case of a new procedure, a location within the procedure repository and a name has to be assigned in a dialog (cf. figure B.11). A path can either be selected by selecting a folder in the tree in the upper part, or inserted manually. New folders can simply be created by specifying the path manually. The name can be arbitrary but must be unique at the given path.
B.5 Planning Perspective

The planning perspective (cf. figure B.12) is the heart of the MPT. It allows to inspect the mission plan, as well as add new timeline entries, change or delete existing ones, to set the planning horizon and to export the sequence of events.
B.5.1 Configure Perspective

General

To properly use the planning perspective it has to be configured by the planning engineer according to his needs. One may resize all different parts, minimize or maximize them, and can even drag them out of the main window e.g. to be displayed on a separate screen. Besides the configuration of the different windows, you probably want to select different timelines to be displayed. Simply tick the corresponding items on the left hand side. Clocks are shown in the Time part. They appear there in the order, they were ticked in the selection window. Similarly to the clocks, organisational units and procedures can be selected in the respective tab and appear in the Master Timeline (again in the order they were selected). Please note, that in the tree of the procedure repository only procedures, but no folders can be selected although a tick-box is displayed for both (cf. figure B.13).

![Figure B.13: Folders can not be selected to be shown in a timeline.](image)

Clocks

The top panel features three clocks. The UTC time, a local clock and the mission elapsed time. To configure which times are displayed here refer to \[B.3.3\].

Link Timeline

The Master Timeline can either be linked to the clocks of the Time part as shown in figure B.14a or remain independent (cf. figure B.14b). If independent, it features its own time axis and scroll bar, otherwise the separate time axis and scroll bar are hidden to spare screen space and it scrolls and zooms (cf. B.5.1) simultaneously with the clocks of the Time part.
Zoom

Using the corresponding buttons, the timelines can be zoomed in or out to several levels, to show a smaller or larger portion of the planning horizon. When the Master Timeline is coupled to the clocks it zooms simultaneously with the clocks and its separate buttons are disabled. The scroll bars scale with the zoom level. Once the complete planning horizon is shown, one can not further zoom out. The highest zoom level (most detailed) shows a portion of 10 seconds of the mission plan. The timelines zoom around the center, i.e. the center remains the same, while the portion of time shown around shrinks or grows.

Scroll to Now & Auto Scroll

The timelines feature a red line indicating the current time. When clicking the Scroll to Now-button the clocks (and if it is coupled also the Master Timeline) jump to the current time. When the auto-scroll function is activated, the clocks automatically
scroll so that the red line stays on a fixed position in the plots while the timeline itself traverses as time is passing (cf. figure B.15). The scroll bar is disabled when auto-scrolling is enabled. The position of the red line (i.e. being centered, or showing e.g. more of the future or past) in the plots can be configured in the preferences (cf. B.3.1). Those functions of course only work properly if the current time lays within the planning horizon.

Figure B.15: Auto-scrolling of the timelines.

### B.5.2 Inspect Timeline

The MPT provides different features to inspect the timelines and the planned timeline entries. To do so, first of all the perspective shall be properly configured, i.e. the relevant timelines selected for display, scrolled to the time of interest and zoomed to an appropriate level as explained before. Already existing timeline entries are shown on the corresponding timeline as yellow bars whose width represent the duration of the timeline entry. Note, that a timeline entry not only appears on the timeline of the underlying procedure, but also on all timelines of the organisational units the procedure occupies. In figure B.16 this is the case for the timeline entry "Snapshot" which appears both on the timeline for the procedure "Snapshot" but also on the one of the organisational unit "ASAP-L".

The labels of the timelines are marked yellow when at least one timeline entry lies within the currently shown range of time. This is especially helpful for low zoom levels (showing large portions of time), as one might miss short timeline entries which are consequently only few pixels wide. The tooltip of the label shows the full name of the timeline (as it might be cropped for large names) and the exact number of timeline entries shown in the current range (cf. figure B.17).

The boxes of the timeline entries contain basically all relevant data of the timeline entry. However, if several timelines are shown and the zoom level is low or the entry quite short, this information gets cropped both in height and width. Therefore, the most important information, the name and times of the timeline entry, are also shown
B.5.3 Add, Change & Delete Timeline Entries

A new timeline entry can simply be added via the button on the top of the planning perspective or via File → New Timeline Entry. A dialog opens and at first you are asked to select the underlying procedure from the tree for the timeline entry. Please make sure, to select a procedure but not a folder, before continuing with Next.

Another dialog window opens (cf. figure 3.18) where you can set the details of the timeline entry. Start and end times are to be specified as UTC times. The default values are the current time for the start and the end is set to satisfy the minimal duration. The minimal and maximal duration (0 representing positive infinity) are displayed as a hint to enter valid start and end dates. The timeline entry can be marked as "immediate" by ticking the corresponding box. This affects how the timeline entry is dealt with during the export of the SoE (cf. B.5.4). If the underlying procedure is already marked as "immediate" (cf. B.4.2) this option is inherited from the procedure and locked. If the underlying procedure uses procedure parameters you have to assign a value of the correct type in the table for every parameter. The insertion of the timeline entry can be finished with Save or be cancelled.

To change an existing timeline entry, simply double-click on it or right-click and select Change.... The very same details dialog opens and you can change the settings.
B.5. Planning Perspective

for the timeline entry.

To delete an existing timeline entry, access its context menu with a right-click on it and select Delete. After accepting the confirmation dialog the timeline entry will be deleted.

B.5.4 Export SoE

The export of the sequence of events is invoked by clicking the corresponding button on the top right of the planning perspective or via File \(\rightarrow\) Export SoE and a dialog window opens (cf. figure B.19).

The range of the mission plan which shall be exported can be specified as UTC date
in the fields "From:" and "Until:". It is recommended to use the preset values. The preset start date refers to the last date until which the last export is valid and the end of the planning horizon is set as the end of the export. The user may specify any dates (the end must be after the start), but if other dates than the preset ones are set, the user is fully responsible to guarantee that the union of all export files resembles a correct, complete and especially continuous mission plan.

Furthermore, the path where the exported files will be stored, can be set. By default it is an subfolder of the root path (cf. B.3.1), but any absolute path can be inserted. By clicking the button labelled "..." you can browse your file system. A comment, which will be added as prefix to the file names may be specified as well. The rest of the file name follows a standardized pattern, identifying the type of the file (see below), the start and end of the export file as well as the date of the file creation.

Last but not least, you may select which files shall be generated (at least one has to be selected).

**Timetagged Telecommand Sequence** This is the timetagged sequence of all telecommands of all timeline entries in the specified time range. It is exported in the file format of the telecommand client and can directly be imported into it. It only contains the information of the timetagged telecommands, i.e. it explicitly does not contain further information of the steps (like the description). Additionally, it excludes steps which have no telecommand attached, as well as steps of timeline entries which are marked to be conducted in an immediate manner.

**Immediate Telecommand Sequence** Similar to the Timetagged Telecommand Sequence it contains only the information of telecommands in the file format of the telecommand client. In contrast to the export file of the timetagged timeline entries, it only contains the steps of timeline entries marked as immediate (of those only the steps with attached telecommand), but none of the timeline entries which are to be conducted in a timetagged manner. As the telecommand client handles telecommands without a timestamp as immediate ones, the timing information which is available in the MPT is excluded as well. To properly conduct the timeline entry, the operator thus has to either refer to the Full Operator Sequence or the SoE Monitoring Perspective of the MPT.

**Full Operator Sequence** This file contains all information available in the MPT of all steps of all timeline entries of the specified range of time. It is exported as a comma-separated file and can thus easily be imported to other software tools (e.g. spreadsheet tools) to display, further format or print the sequence of events.
B.5.5 Set Planning Horizon

The planning horizon is the time window which is relevant for the planning of the immediate future. It commonly ranges from a few weeks (1-3) in the past to some weeks (2-6) in the future and is shifted regularly forward in time as time passes. This is to be done by clicking the Set Horizon-button on the top right of the planning perspective. A simple dialog window opens (cf. figure B.20), where the start and end dates (UTC) of the planning horizon can be set. The preset values are the ones currently set. The start can be set arbitrarily, while the end has to be after the start and must be in the future. By clicking Set the dialog is confirmed and closed.

As time passes, more and more timeline entries will accumulate in the mission plan, while the ones in the far past are no longer of relevance for the further mission planning. To prevent that the MPT runs slow due to the amount of actually useless timeline entries (which have to be loaded at system start-up, be regarded when the timelines are drawn etc.), from time to time one should clean the mission plan when moving the planning horizon forward. Set and Clean removes timeline entries, which lie outside of the specified planning horizon, and archives them to a special XML-file. Thus, they are no longer available in the MPT and do not obstruct its performance, but are kept for documentation purposes.

Figure B.20: Set Planning Horizon Dialog.

B.5.6 Other Features of the Planning Perspective

Further Information Panels The planning perspective features some more information panels on the top right (cf. figure B.12). The two fields grouped with Planning Horizon display the start and end of the current horizon. The end date is color coded. It is green, if the horizon extends sufficiently far into the future (to set the limit confer to B.3.2). If this limit is violated, but the horizon end is still in the future, it is displayed yellow to warn the planning engineer about this. If the horizon end is even in the past, the display turns red.

The date until when the sequence of events was exported is displayed similarly. If
the export extends sufficiently far into the future (see B.3.2 to set the limit) it is
green; yellow if the limit is violated but the end of the export still in the future; and
red otherwise. The export status, indicates whether the export is congruent with the
current mission plan available in the MPT. If this is the case, the status is green and
"up-to-date". If parts of the plan, which were already exported, are changed, it turns
red and states "outdated" to warn the planning engineer about this.

**Lock Controls** Via the main menu Window → Lock Controls all controls of the
GUI except from the lock itself, the scrolling of the timelines, and displaying the
details of a timeline entry, can be locked. This shall prevent unintended user input
when the MPT is used to monitor ongoing activities.

**Safe the Mission Plan** If the mission plan was changed, i.e. any timeline entry
added, deleted or changed, the horizon set, or the SoE exported, this is indicated
with an * on the Master Timeline. The changes can be saved if the Master Timeline
is selected via File → Save or Ctrl+S.

### B.6 SoE Monitor Perspective

The SoE monitor perspective (cf. figure B.21) is rather simple. Similarly to the
planning perspective, it features clocks on the top (to configure them see B.3.3). The
main part of the perspective is occupied by the SoE Table. This table shows all steps
of the current plan with all their details in the order of time. The column "Until"
displays a counter to the start of the step, "Procedure" the name of the underlying
procedure, "# in Proc" the index of the step within the procedure, and "isImmediate"
indicates whether this step is to be performed in immediate manner or not. The other
columns hold the very same information as defined in the procedure, while relative
times are converted into absolute ones and placeholders of procedure parameters
replaced with their assigned value. The columns can be resized and rearranged to
the user's needs.

Steps which are in the past are displayed grey and italic; steps which are currently
ongoing are highlighted yellow and bold; and future steps are not highlighted at all.
If the auto-scroll function is enabled (top right corner), the algorithm tries to always
reveal the currently ongoing steps as well as at least three future and three past steps
(ordered in this priority if not enough space is available to reveal all of them).

There is no possibility (except from the main menu) to change the mission plan or
procedures from this perspective.
Figure B.21: SoE Monitor Perspective.
This page is intentionally left blank.
Appendix C

Index of Source DVD

All contents of the attached DVD are also included in the projects SVN-repository, which is stored on servers of the Professorship of Space Technology of Julius-Maximilians University of Würzburg. Please contact Prof. Dr. Hakan Kayal¹ if you would like to access the SVN-repository.

Documentation

The folder Documentation of the attached DVD contains the following files.

<table>
<thead>
<tr>
<th>File Name</th>
<th>Technical_Documentation_SONATE_MPT.pdf</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Type</td>
<td>Portable Document Format (PDF)</td>
</tr>
<tr>
<td>Description</td>
<td>The technical documentation of the MPT generated by doxygen (cf. 7.6). This includes descriptions of all Java packages and classes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Name</th>
<th>Technical_Documentation_SONATE_MPT.html</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Type</td>
<td>Hypertext Markup Language file (HTML)</td>
</tr>
<tr>
<td>Description</td>
<td>A HTML version of the technical documentation of the MPT generated by doxygen (cf. 7.6). This includes descriptions of all Java packages and classes and is represented nicely on several html pages. The user shall access the html pages (which are internally stored in the subfolder doxygen) only through this file.</td>
</tr>
</tbody>
</table>

¹kayal@informatik.uni-wuerzburg.de
Software Product

The folder Software_Product of the attached DVD contains the following files.

<table>
<thead>
<tr>
<th>File Name:</th>
<th>SONATE_MPT_win32.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Type:</td>
<td>ZIP-archive</td>
</tr>
<tr>
<td>Description:</td>
<td>The final software product of the MPT delivered with this thesis for the Windows 32-bit platform. For installation details refer to appendix B.2.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File Name:</th>
<th>SONATE_MPT_win64.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Type:</td>
<td>ZIP-archive</td>
</tr>
<tr>
<td>Description:</td>
<td>The final software product of the MPT delivered with this thesis for the Windows 64-bit platform. For installation details refer to appendix B.2.</td>
</tr>
</tbody>
</table>

Software Sources

The folder Software_Sources of the attached DVD contains the following files.

<table>
<thead>
<tr>
<th>File Name:</th>
<th>SONATE_MPT_sources.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Type:</td>
<td>ZIP-archive</td>
</tr>
<tr>
<td>Description:</td>
<td>The source files of the final software product of the MPT. Besides the .java class-files stored according to the package structure, the archive also includes the used logos as well as all files needed by the RCP application: Application Model (sonate.mpt.rcp.core.e4xmi), product-file (sonate.mpt.rcp.product), plugin.xml, build.properties, MANIFEST.MF. External libraries are not included!</td>
</tr>
</tbody>
</table>
List of Figures

4.1 System context diagram ................................................. 14
4.2 Use case diagram for the systems engineer ......................... 15
4.3 Use case diagram for the planning engineer ........................ 16
4.4 Use case diagram for the operator .................................. 17
4.5 Class diagram for subject specific classes ......................... 20
4.6 Pencil sketch of the GUI prototype of the planning perspective 23
4.7 Pencil sketch of the GUI prototype of the procedure editor . . 24
4.8 Pencil sketch of the GUI prototype of the monitoring perspective 25
5.1 Example for Safe Planning ............................................. 29
5.2 Example for Sigma Resource Propagation ............................ 31
5.3 Example for Sigma Resource Propagation. Case 1 .................. 32
5.4 Example for Sigma Resource Propagation. Case 2 ................. 33
5.5 Example of Finite State Machine ..................................... 34
5.6 Sigma Resource Propagation of a Finite State Resource ......... 34
6.1 Diagram of the system architecture .................................. 38
6.2 Example for parametrizing a procedure ......................... 47
7.1 Screenshot of the actual planning perspective ..................... 60
7.2 Screenshot of the actual procedure editing perspective ........ 61
7.3 Screenshot of the actual SoE monitor perspective .................. 61
B.1 Main menu and perspective switch ................................ 89
B.2 Binary directory of the MPT ......................................... 90
B.3 Overview of the Preferences Dialog ................................ 91
B.4 Different settings for the fraction for the "now-line" .......... 93
B.5 Preferences Dialog. Setting the App Title ...................... 93
B.6 Preferences Dialog. Mission tab ................................... 94
B.7 Preferences Dialog. Clocks tab ..................................... 95
B.8 Overview of the Procedure Editing Perspective ................ 96
B.9 Sequence of Events tab of the procedure editor ............... 98
B.10 Title of a changed procedure ..................................... 100
B.11 Save dialog for a new procedure .................................. 100
B.12 Overview of the Planning Perspective .............................. 100
B.13 Folders can not be selected to be shown in a timeline ........ 101
B.14 Linking the Master Timeline to the Time part .................. 102
**List of Tables**

8.1 Verification matrix for operational requirements ............................................. 65
8.2 Verification matrix for design requirements ..................................................... 66
8.3 Verification matrix for performance requirements .............................................. 67
8.4 Verification matrix for functional requirements ................................................ 67
8.5 Verification matrix for functional requirements (continued) ............................. 68

**List of Listings**

7.1 Example for model class prepared for JFace databindings ............................... 53
7.2 Example for databinding initialisation ............................................................ 53
7.3 Example for model class prepared for JAXB ................................................... 55
7.4 JAXB marshalling ConcreteModelObject (cf. 7.3) ........................................ 55
7.5 JAXB resulting XML-file for a ConcreteModelObject (cf. 7.4) ......................... 56
7.6 JAXB unmarshalling ConcreteModelObject (cf. 7.5) ....................................... 56


In 14th International Conference on Space Operations (Space Ops 2016), Daejeon, Republic of Korea. OCLC: 931086945.