Interdisciplinary Project Report - Organization and Development of the Mission Operations System for the MOVE-II CubeSat

Technical Report · March 2018

DOI: 10.13140/RG.2.2.35355.36646

CITATIONS
0

READS
50

1 author:

Alexander Lill
Technische Universität München
5 PUBLICATIONS **4** CITATIONS

SEE PROFILE

Chair of Astronautics
Prof. Prof. h.c. Dr. Dr. h.c.
Ulrich Walter

Interdisciplinary Project

Organization and Development of the Mission Operations System for the MOVE-II CubeSat

RT-IDP 2016/05

Author:
Alexander Lill

Supervisor: Dipl.-Ing. Martin Langer
Chair of Astronautics
Technische Universität München

Zusammenfassung

Dieser Bericht beschreibt meine Aktivitäten während meines Interdisziplinären Projekts (IDPs) am Lehrstuhl für Raumfahrttechnik an der Fakultät für Maschinenwesen der Technischen Universität München.

Während meines IDPs arbeitete ich am Operations System, welches für den Missionsbetrieb des CubeSats MOVE-II verwendet wird. Zu meinen Aufgaben gehörte die Gründung des Operations Teams, die Sammlung von Anforderungen für das zu entwickelnde System, die Erstellung von Skizzen der Benutzeroberfläche und die Definition der gesamten Softwarearchitektur. Gleichzeitig übernahm ich die Koordination der Entwicklung des Systems und die Organisation des Teams. Mit einer Beschreibung der finalen Softwarearchitektur beende ich diesen Teil meines IDPs.

Zusätzlich zur Entwicklung des Operations Systems entwickelte ich ein Tool zur halbautomatischen Erfassung von Fehler-Reports namens “Elfriede”. Dieser Bericht beschreibt die aufgetretenen Probleme in unserem Projekt und wie diese beseitigt werden konnten. Dieser Teil meines IDPs endet mit einer Erläuterung der Implementierung von “Elfriede”.

Abstract

This report describes my activities during my Interdisciplinary Project (IDP) at the Chair of Astronautics at the Department of Mechanical Engineering, Technical University of Munich (TUM).

During my IDP I worked on the Operations System that will be used during Mission Operations of the CubeSat MOVE-II. My work included founding the new Operations Team, the collection of requirements for the system that shall be developed, the creation of Mock-Ups for the user interface and the definition of the overall software architecture. In parallel to that I coordinated all development efforts and organized the team. This part of my IDP concludes with the description of the final software architecture of the Operations System.

Furthermore I developed a semi-automated bug-tracking tool called “Elfriede”. This report provides a problem statement describing the issues we experience in our project as well as the solution that was found. This part concludes with a description of the implementation of “Elfriede”.

Contents

List of Figures

Acronyms

ADCS Attitude Determination and Control System

API Application Programming Interface

AUTH Authentication

BPO Beacon Poster

BPR Beacon Processor

CDH Command and Data Handling

CMD Commanding

COM Communication

DLR Deutsches Zentrum für Luft- und Raumfahrt

EPS Electrical Power System

ESPACE Earth Oriented Space Science and Technology

HK Housekeeping

HORST Humble On-board Reconfiguration State Transformer

HPR Housekeeping Processor

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDP Interdisciplinary Project

JSON JavaScript Object Notation

LRT Lehrstuhl für Raumfahrtechnik

MOVE-II Munich Orbital Verification Experiment II

NOTS Notification Service

OAuth2 Open Authorization 2

OPS Operations

PL Payload

REST REpresentational State Transfer

STATS Statistics Service

STR Structure

TCP Transmission Control Protocol

THM Thermal

TUM Technische Universität München

WARR Wissenschaftliche Arbeitsgemeinschaft für Raketentechnik und Raumfahrt

1 Introduction

The Scientific Workgroup for Rocketry and Spaceflight [1] (Wissenschaftliche Arbeitsgemeinschaft für Raketentechnik und Raumfahrt (WARR)) is a student organization at the Technische Universität München (TUM) Department of Mechanical Engineering and located at the Chair of Astronautics (Lehrstuhl für Raumfahrtechnik (LRT)).

This scientific workgroup is concerned with satellite technology, space elevators, autonomous rovers and of course rocketry and is also known for its Hyperloop[2] team.

One of the biggest groups of students is part of the satellite technology project MOVE (the *Munich Orbital Verification Experiment*). The current mission is MOVE-II, a one unit CubeSat with dimensions of 10x10x10cm and a maximum weight of 1.33kg. It is the second satellite of the TUM and the follow-up project of First-MOVE.

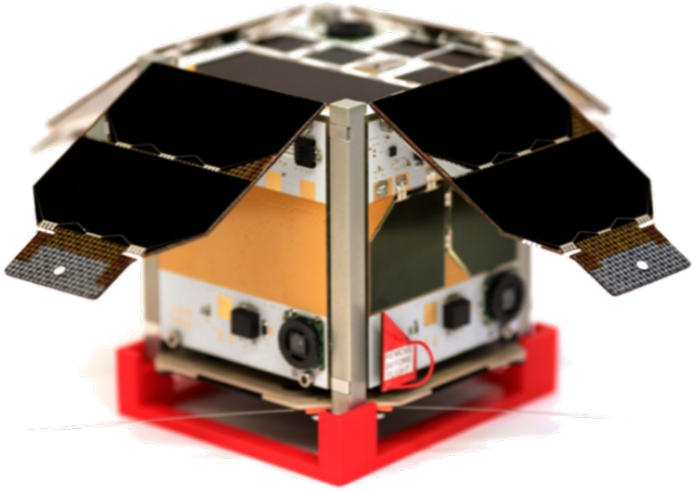

Fig. 1–1: MOVE-II Satellite.

Figure 1–1 shows the MOVE-II CubeSat. The MOVE-II mission has the following goals:

- Build a reusable satellite bus
- Test our payload, prototypes of new solar cells
- Educate students about how to build satellites
- Train students in how to operate satellites

The satellite is developed in cooperation with the LRT and the WARR and is partly funded by the German Aerospace Center (Deutsches Zentrum für Luft- und Raumfahrt (DLR)).

[1] http://www.warr.de
[2] http://hyperloop.warr.de/

The MOVE-II project can be defined as a system of systems as shown in Figure 1–2 with the following parts:

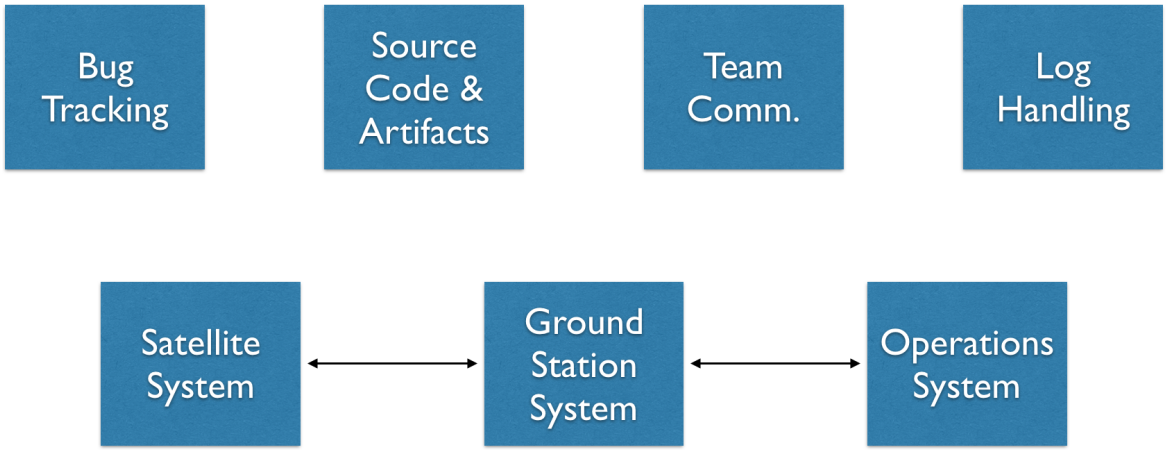

Fig. 1–2: System of Systems.

Satellite System The Satellite System consists of hardware and software from the respective subsystems:

- Structure (STR): Provides the surrounding structure for all other subsystems including the deployable solar panels and the deployment mechanism.
- Electrical Power System (EPS): Stores and distributes the power for all other subsystems.
- Communication (COM): Enables communication between the satellite and the ground station.
- Command and Data Handling (CDH): Handles commands from the ground station and stores all data generated on the satellite.
- Attitude Determination and Control System (ADCS): Allows to control the satellite's attitude.
- Payload (PL): Provides the means to determine the performance of new solar cells.
- Thermal (THM): Measures the thermal attributes of the satellite.

Ground Station System The Ground Station System consists of the hardware required for receiving and transmitting signals from and to the satellite and all necessary hardware and software to decode and encode these signals.

Operations System The Operations System consists of all the hardware and software necessary for analyzing the satellite's data and controlling the satellite and will be explained in more detail in the following chapter.

Bug Tracking The Bug Tracking component is a separate software that is used to keep track of issues with the Satellite, Ground Station and Operations System.

Source Code & Artifacts The Source Code & Artifacts component is a separate software that is used to store and version all source code and provide all necessary artifacts for the other systems.

Team Communication The Team Communication component is a separate software that is used to enable real-time communication between team members and groups of team members and will be explained in more detail in chapter 3 "Semi-Automated Bug-Tracking with Elfriede".

Log Handling The Log Handling component stores all the logs generated by software running in the Ground Station and Operations System.

One of the main parts of my Interdisciplinary Project at the LRT was the definition of the overall software architecture and all involved components in the Operations System and the development of these components (see chapter 2). I additionally worked on parts of the other systems. One of these other tasks included the creation of the semi-automated bug-tracking tool "Elfriede" (see chapter 3). This report ends with a conclusion of my now almost two years of interdisciplinary work at the Chair of Astronautics in chapter 4.

2 MOVE-II Operations System Software

The Operations System is an important and complex part of the whole mission. Many different functionalities have to be implemented, and many different interfaces have to be covered. As it is not feasible to deal with this alone the Operations team was founded. This new team had the tasks to collect the high level requirements for this system and create mock-ups of the user interface. Furthermore the software architecture had to be defined and successively implemented. The resulting software architecture is explained in more detail in the last section of this chapter.

2.1 Timeline

- 2016-05-04: Joined the MOVE-II COM subsystem to work on custom ground station software
- 2016-10-24: First meetings regarding the new Operations subsystem
- 2016-11-02: Founding of the OPS subsystem
- 2016-11: Creation of the first requirements for OPS
- 2016-12: Creation of the first mock-ups, first drafts of the architecture
- 2017-01: Start of the implementation of the first microservice (Logbook)
- 2017-02: Start of the implementation of the user interface and the second microservice (EPS)
- 2017-03: Start of the implementation of the third, fourth and fifth microservice (PL, THM, CDH) and the Housekeeping Processor (HPR)
- 2017-04: Start of the implementation of the beacon processor
- 2017-05: First usage of the Operations System for the Thermal Vacuum Chamber Tests, sixth and seventh microservice (ADCS, COM)
- 2017-08: Start of the implementation of the health microservice and notification service
- 2017-10: Start of the implementation of the authentication microservice and statistics service
- 2018-01: Start of the implementation of the Housekeeping (HK) microservice

2.2 Founding of the Operations Team

New team members for the OPS team had to be found. For this the primary source was a quick introduction of the MOVE-II project in the microprocessors lecture and a Kickoff Event organized for the whole MOVE-II project. A weekly meeting was introduced right from the beginning. As many of the newly acquired team members were studying in the international Master's Program *Earth Oriented Space Science and Technology (ESPACE)* lots of meetings took place at the main campus to reduce overall traveling time of the team members.

The composition of the OPS team over time is shown in Figure 2–1. It shows how many members the OPS team had (gray line) and if they were classified as developers (blue) or non-developers (orange). It is clearly visible that the number of developers increased after the initial collection of requirements and creation of the mock-ups.

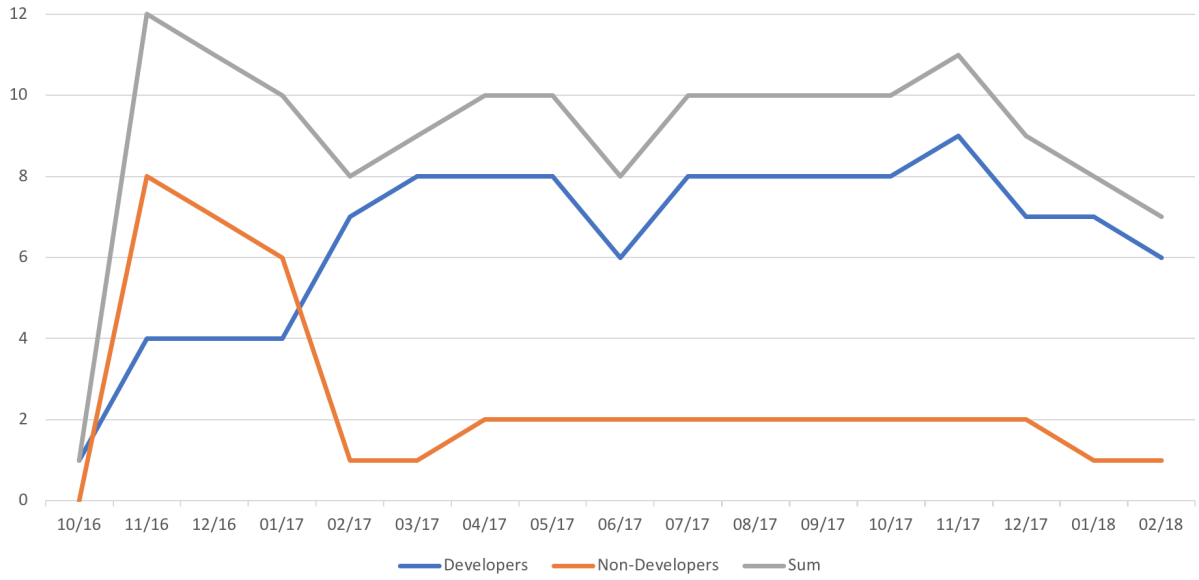

Fig. 2–1: OPS Team Members.

In this phase a first project plan for the new OPS team (see Figure 2–2) was created. This plan gave a first outline of the necessary steps to take and was then successively executed more or less as planned.

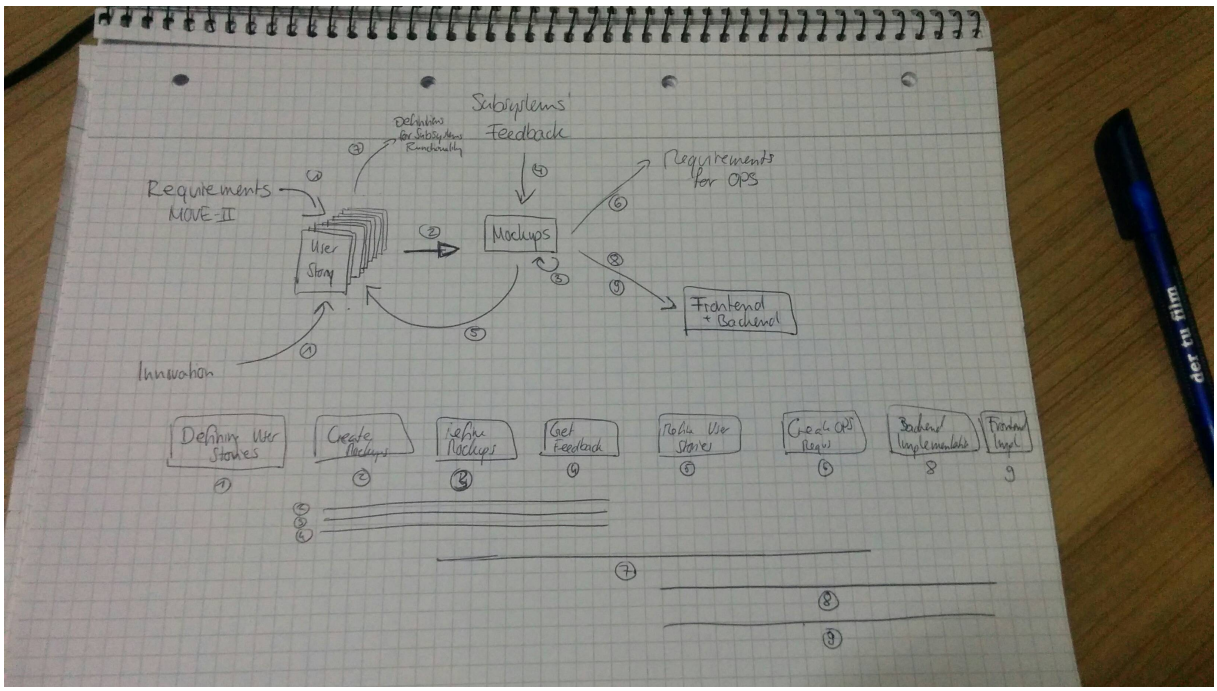

Fig. 2–2: First Project Plan.

2.3 Collection of Requirements

There are about 170 Requirements defined for the MOVE-II mission. These requirements describe the goals of the mission and which functionalities the different subsystems have to provide for this mission. The requirements in Table 2–1 are a selection of our real requirements.

Unfortunately no requirements were available for the OPS subsystem. This resulted in not knowing which goals the OPS subsystem has to reach and which functionalities shall be provided. Therefore this lack of requirements had to be resolved by carefully defining what the OPS subsystem should enable operators to do with the MOVE-II satellite.

In order to obtain the requirements for the OPS subsystem the requirements for all other subsystems were analyzed and used to deduce the new requirements. As all members of the OPS subsystem including myself were not that experienced with spacecraft operations and therefore biased by traditional operations concepts this led to many requirements that were very basic, but at times also quite innovative for this industry. These requirements often resulted in questions for the respective subsystems that had to be clarified, e.g. if certain data is available, or how data and workflows actually look like.

Table 2–2 shows examples of deduced requirements.

Tab. 2–1: Selection of requirements for the MOVE-II CubeSat

ID	Requirement	Rationale
STEX-DLR-01	MOVE-II shall be a One Unit (1U) CubeSat according to the CubeSat Design Specification (CDS).	Since the Multi-Purpose Active-Target Particle Telescope was removed from the design due to insufficient funding by the DLR, MOVE-II is further expected to be a 1U CubeSat.
STEX-LRT-01	MOVE-II shall be designed as a 1U satellite payload platform for future missions.	MOVE-II serves the purpose of proving that the satellite system provides all necessary capabilities for future payload missions. With the MOVE-II mission, flight heritage on this system is gathered. The system will be reused as satellite bus for future missions. This will save time and effort then, plus provide the future engineering team with the reliability of a completely tested satellite bus system.
SYS-01	The MOVE-II satellite shall operate for at least 6 months.	MOVE-II shall be a bus system adaptable to different future payloads. Therefore a minimum runtime of half a year is reasonable. In case of the actual MOVE-II payload, the duration of six months is mandatory to evaluate the degradation of the solar cells.
CDH-07.1	The CDH shall be able to be reprogrammed in off-nominal behavior in orbit.	This gives us the possibility to correct mistakes or reload the image in case it is broken.

Tab. 2–2: Deduced User Stories for the Operations System

Requirement	I want ...	So that ...
The project duration shall be three years, including six months of operations.	to have the system uptime viewable from the groundstation	we know how many hours it has been running since the last reboot
	to see the number of days or hours since the mission started	I can see how long the mission has been going on
All subsystems should provide telemetry.	to see important values for each subsystem (e.g. temperature)	i can check if they are in their defined and allowed ranges
	to be able to set minimum and maximum values for all parameters	I can automate the coloring and warnings for all telemetry
The CDH shall be able to command the satellite in all nominal operation modes, and provide safehold and data collecting measures in case of off-nominal behavior.	I want to see the current mode of the satellite (e.g. safemode / normal)	I see on one glance what is going on
	I want to have the ground station automatically log separately data in case of non nominal behaviour	easier debugging can be done afterwards
	to have automatic sample rate adjustment for non nominal mode	have more detailed data
	to be notified in case of abnormal behavior, e.g. Slack, Mail, SMS	so that I can start preparing the next overpass

These requirements (phrased as *User Stories*, concrete statements that state what a possible user with a certain role wants from the software, and what he hopes to accomplish with this) were then split into different groups. These groups represent either general user stories (that should be implemented by all parts of the software) or for certain areas of the software, e.g. for a screen that is aimed for operators of a certain subsystem, for administrative users or for events like the launch of our satellite.

2.4 Creation of Mock-Ups

After the creation of requirements as User Stories the team started with creating mock-ups of the user interface.

The idea behind this was to:

- have the team think more about the User Stories and how they could be implemented
- improve the User Stories with the knowledge gained from drafting the user interface
- get feedback from the other subsystems by showing them the mock-ups and improving them together with the future users of the interface (instead of just showing them text and asking them if they have any feedback)
- be able to create first prototypes and test them by using mocked screens that we imagined for the mission control center. Knowing which computers, monitors and projectors will be available we also created a preliminary purpose for all of them

The first mock-ups were created in groups using pen and paper. In Figure 2–3 one of the many manual drafts of, in this case, the EPS data visualization can be seen.

Afterwards these mock-ups were digitized as vector graphics using the collaborative online sketching tool ”Figma”[1] to be able to have a more real representation of the future user interface (see Figure 2–4).

2.5 Draft of the Architecture

After it became more and more clear which functionalities the Operations System will have to provide the first draft of a software architecture was created.

As the Operations System has to cover a large spectrum of use cases the complexity of the overall system is quite high. To handle this complexity many best practices and software engineering principles were kept in mind during the draft of the architecture.

The software was modularized at the system level, strictly following a microservice architecture. Therefore the system consists of many different microservices that provide only one single function, taken from the Unix philosophy “Do one thing and do it well”. One example for this is the Thermal (THM) microservice, that stores all temperature data and allows to query this data. The microservice architecture makes the software

[1] `https://www.figma.com`

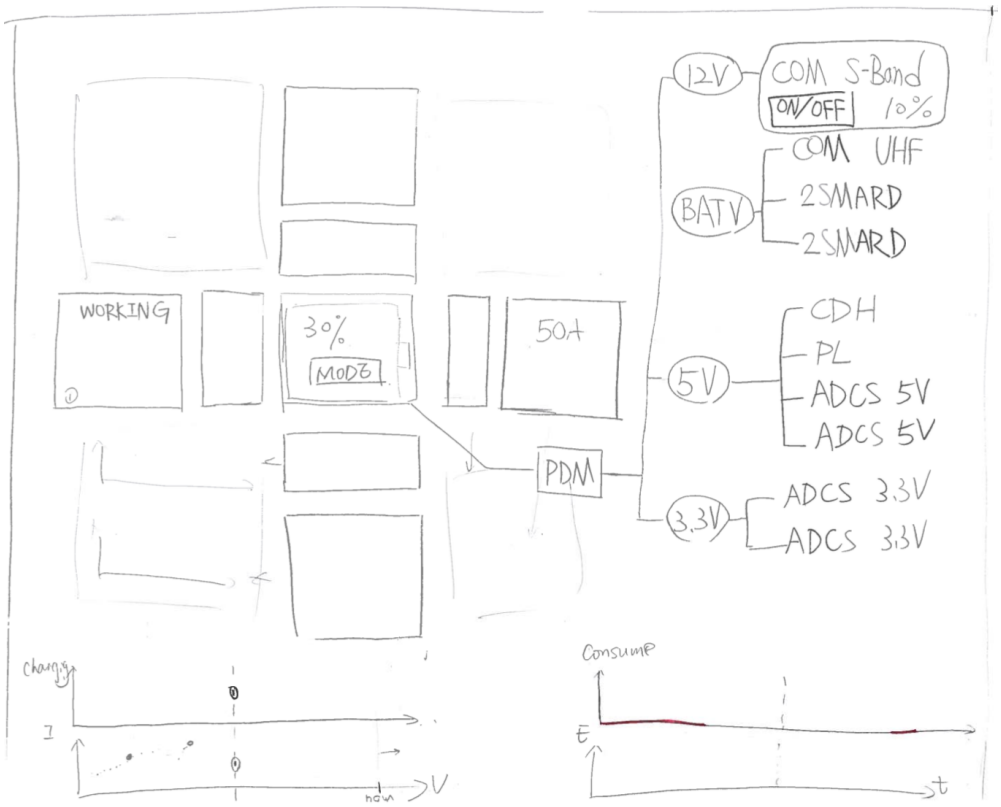

Fig. 2–3: First Mock-Ups.

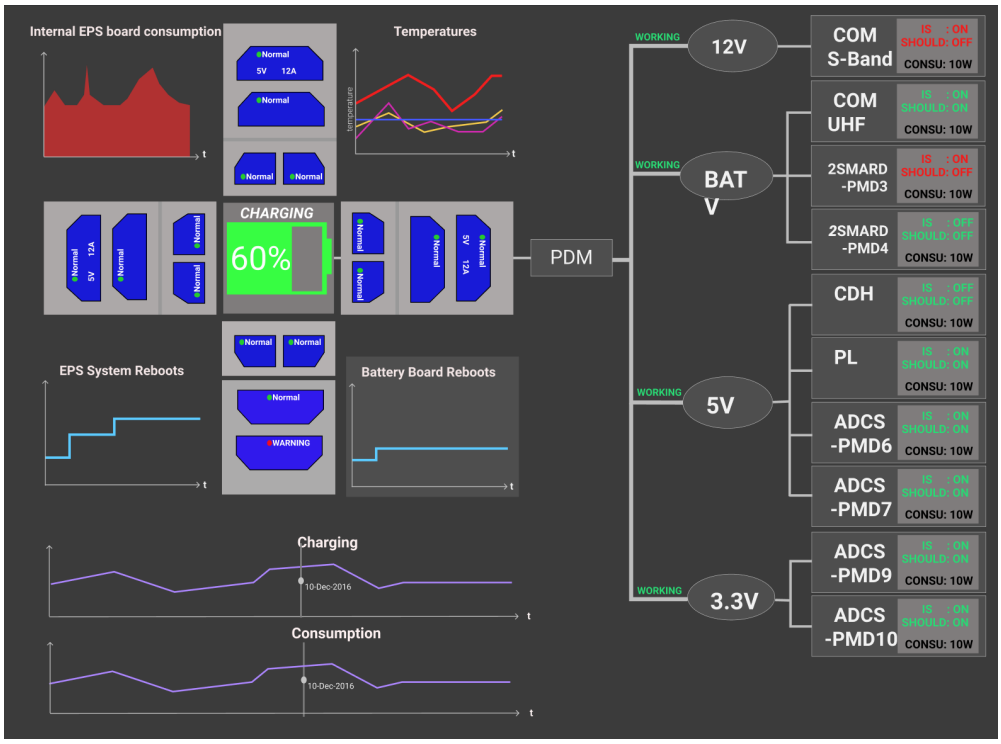

Fig. 2–4: Digital Mock-Ups.

components easier to understand, develop and test and furthermore simplifies continuous deployment through the low coupling between microservices.

For further details about the architecture of the Operations System please refer to the Interdisciplinary Project Report of Alexandru Obada, Chapter 3.

2.6 Development Process

The development process of the Operations System is based on agile software development methodologies and was largely inspired by Scrum. For our development efforts we took the Scrum development process as an inspiration and adjusted it to our needs. This was necessary due to the following reasons:

- All members of the OPS team are students and not working for the project full time. Many of the members work for the MOVE-II project voluntarily or as part of an . This leads to less weekly working hours than for full-time developers and makes physical daily meetings basically impossible.
- The Scrum process often defines the developers as professionals with many years of practical experience and teamwork. As MOVE-II is a student project many members do not have much practical experience or have never worked in such a big team.
- Due to the scarcity of resources it was not possible to map every role of the Scrum process to a separate person.

The Scrum development process consists of artifacts created and used in the process, events that define the process and roles that participants of the process have.

The following paragraphs will briefly describe the most important parts of Scrum and provide a summary of the adjustments we applied. For further information about the standard Scrum process please refer to more detailed descriptions e.g. on the website ScrumAlliance.org [2].

2.6.1 Artifacts

Product Backlog The *Product Backlog* is an ordered list of everything known to be needed for the product. This list contains all User Stories that were imagined for the product. This list is ordered from the highest priority to the lowest priority.

Sprint Backlog The *Sprint Backlog* is a subset of the User Stories contained in the Product Backlog. These User Stories were selected to be implemented in the current Sprint. All the contained User Stories were estimated regarding their effort by the team before the Sprint starts.

Product Increment A product increment is a new version of the developed product that was improved since the last version. These improvements might be new features, fixed bugs or improved performance, reliability and so on.

[2] https://www.scrumalliance.org/why-scrum/scrum-guide

2.6.2 Events

The Sprint The *Sprint* is a time-box of a certain length that is used to split the Scrum process into phases. Sprints normally have consistent durations during a project and per definition end in a potentially shippable product increment. The Sprint Backlog is normally not changed during a Sprint to avoid disturbances like frequently changed user stories or requirements from the team.

Adjustments: The duration of a Sprint is generally set to a length of two to four weeks. In our environment the duration was shortened to one week. Experience shows that for longer Sprints work was only done in the few days before the Sprint ends, and that the amount of finished work can be increased by focusing more on short-term goals that can be reached during shorter sessions of a few hours, for example in the evenings or the weekends.

Sprint Planning At the end of every Sprint (and before the beginning of the next Sprint) the team collaboratively decides on the User Stories from the Product Backlog that should be part of the next Sprint Backlog. In the Sprint Planning the goals for the next Product Increment are set and the effort to fulfill these goals is estimated.

Adjustments: The estimation of tasks was not formalized as normally done in the Scrum process (e.g. via *Planning Poker*, where the whole team estimates all tasks together), but was done more briefly during the Sprint Planning event to reduce meeting time. The tasks were mostly estimated by the team leader and the estimate was confirmed by the rest of the team. Raised objections were discussed in more detail and estimations consecutively adjusted.

Daily Scrum *Daily Scrum* describes the daily meeting of the development team to quickly discuss their progress since the day before, what they are planning to do until tomorrow, and which impediments they currently see in getting their work done. It is normally very short (up to 15 minutes) and should take place at the same time and location every day.

Adjustments: Daily meetings were completely removed from the process due to the voluntary nature of the project and the fact that there were rarely times where all team members were free for a physical meeting. In the beginning of the project daily virtual meetings were tried out. Here each team member was asked to provide the following information via our team communication tool *Slack*: “What has been done the day before?”, “What is planned to be done today?” and “Which impediments are there?”. Experience shows that the voluntary team members rarely work on the project every day, but rather in specific time frames like for example weekends or on some evenings every week. Therefore the daily meetings were changed to an on-demand approach, most often implemented by chat messages on *Slack*.

Sprint Review The *Sprint Review* is held at the end of every Sprint to inspect the latest Product Increment and update the Product Backlog if necessary.

Sprint Retrospective The *Sprint Retrospective* is an event where the team reflects on the past Sprint and analyzes its own work to deduce what went well and in which areas there might be potential for improvement. Using this information a plan is created how these lessons learned can be implemented.

2.6.3 Roles

Product Owner The Product Owner is responsible for the Product Backlog. User Stories need to be clearly defined and ordered by priority according to the Product Owner's vision of the product. The Product Owner is accountable for the delivered product.

Adjustments: See *Scrum Master*

Scrum Master The Scrum Master's responsibility is to make sure that the Scrum process is followed and that all rules, practices and values of the process are understood. The Scrum Master is a servant-leader to the Product Owner and the team of developers and also tries to remove impediments to the team's progress.

Adjustments: Due to the limited number of available persons in the project the same person occupied the roles of Product Owner and Scrum Master. Here my role as Team Leader was extended to not only prioritize the User Stories in the Product Backlog and ensure proper understanding of these items, but also to remove impediments from my team and ensure that the agile software development approach is understood. This also included the coordination and communication with the other team members from the other subsystems and hopefully allowed my team members to focus on their development activities.

Developer The Developers implement the different User Stories that are contained in the Sprint Backlog. The developers are encouraged to organize and manage their own work.

2.7 Operations System Software Architecture

After almost a year of iterative development of the Operations System the software architecture is now stable and will probably not change much anymore. Figure 2–5 shows the current architecture and will be explained in the subsequent paragraphs.

As mentioned in the chapter "Draft of the Architecture" the Operations System follows a microservice architecture. Figure 2–5 shows all the different components of the Operations System which will be consecutively explained in the following paragraphs.

Figure 2–6 shows all microservices, that store and retrieve data from the database.

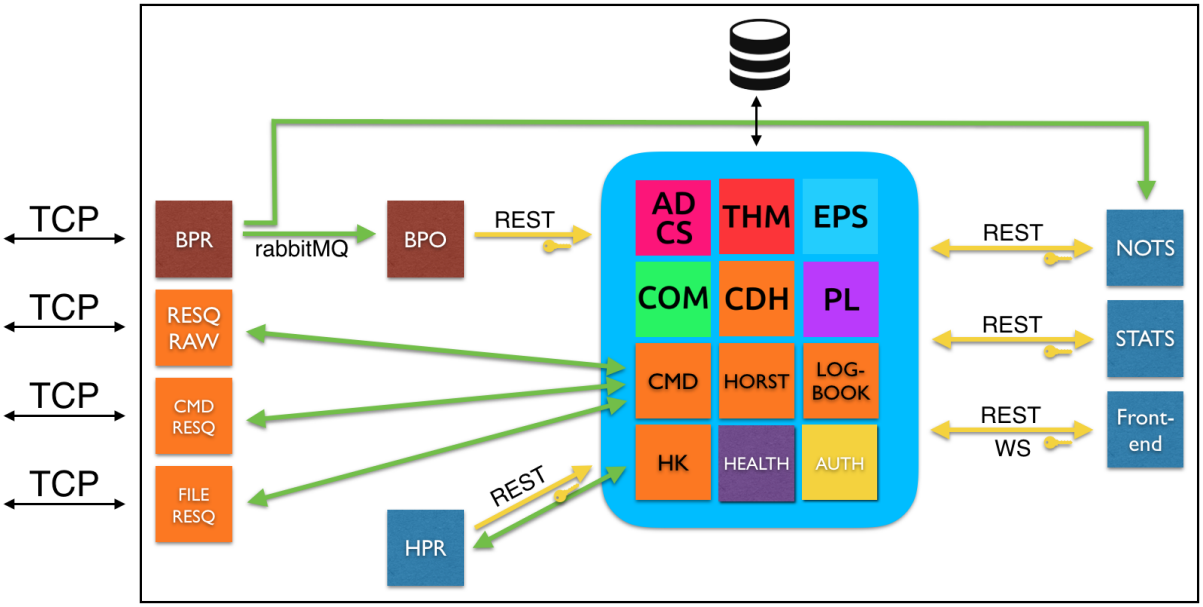

Fig. 2–5: OPS Software Architecture.

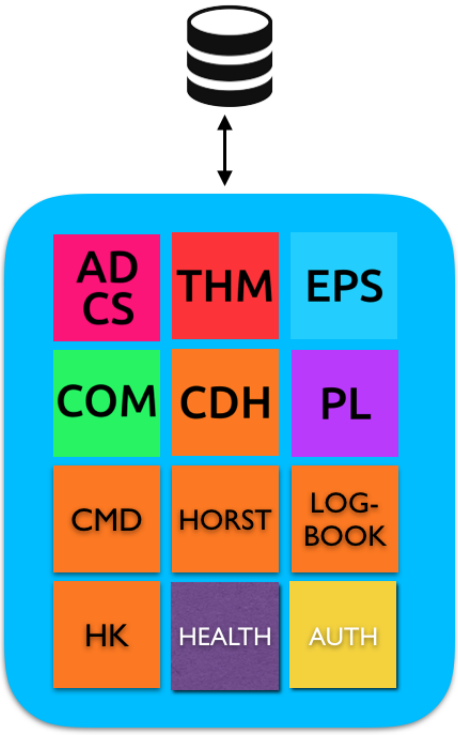

Fig. 2–6: OPS Microservices.

First of all you can see that every subsystem has its own microservice.

- The Thermal (THM) Microservice handles all temperatures that are received from the satellite, and stores them into its own table in the database.
- The Electrical Power System (EPS) microservice handles all current, voltages and other telemetry data received from the satellite that is concerned with the EPS subsystem on board of the satellite.
- The same is true for all microservices in the upper half of the light blue box.
- The responsibility of the Commanding (CMD) microservice is to handle all commanding that we can do with our satellite. This includes for example sending commands and receiving their results or up- and downloading files.
- The Humble On-board Reconfiguration State Transformer (HORST) microservice handles all data that is generated by HORST on the satellite, and is separated from the CDH microservices to increase the performance of this important data source.
- The Logbook microservice handles all downloaded journald logs and allows to manage alerts. These alerts can be compared to Redmine tickets and can be used to track anomalies of the Satellite System.
- The HK microservice handles imports of downloaded housekeeping files.
- The Health microservice handles all health data generated by all the other microservices, stores them into the database and allows to query this information.
- The Authentication (AUTH) microservice regulates the access to all the microservies and ensures that only authenticated users can retrieve and store data from / to the Operations System using the Open Authorization 2 (OAuth2) protocol.
- All microservies provide an interface for storing and retrieving data using the REST technology that is built upon simple Hypertext Transfer Protocol (HTTP) methods. More information regarding this topic can be found in the Interdisciplinary Project (IDP) report of Thomas Zwickl.
- For communication that should take place asynchronously, for example using queues and load sharing algorithms, we use the open-source messaging broker rabbitMQ. More information regarding this topic can be found in the IDP reports of Alexandru Obada, Cristian Soare and Constantin Costescu.

All components that handle communication from and to the satellite are shown in Figure 2–7. These components are connected to the ground station and receive data from there after being decoded (coming from the satellite) or can send data there (which will be sent to the satellite after being encoded).

- The Beacon Processor (BPR) parses the raw Beacon Information which is received as an array of bytes. It is then interpreted and sent in an intermediary representation to the Beacon Poster (BPO) using rabbitMQ. The Beacon Poster transforms the received data into the widely used JavaScript Object Notation

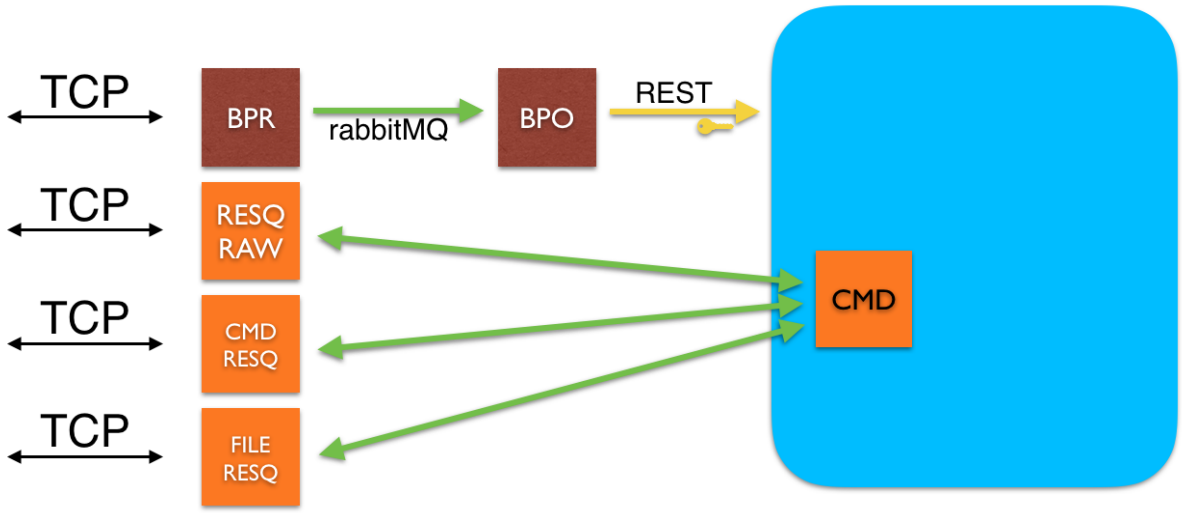

Fig. 2–7: Communication Components.

(JSON) format and sends this to the different microservices using the Hypertext Transfer Protocol Secure (HTTPS) REST interface. More information regarding the parsing and posting can be found in the IDP report of Constantin Costescu.

- The different RESQ boxes talk to their respective counterparts on the satellite. The RESQ components allow to send commands to the satellite and retrieve their results or up- and download files. These components directly talk to the ground station via Transmission Control Protocol (TCP) sockets on the one side, and the Commanding (CMD) microservice on the other side via rabbitMQ.

In Figure 2–8 you can see the HPR. The HPR receives requests via rabbitMQ from the HK microservice to import housekeeping files. These files have to be downloaded from the satellite before. The HPR then sends back progress updates using rabbitMQ while, at the same time, transmitting the parsed data to the respective microservices using the HTTPS REST interface.

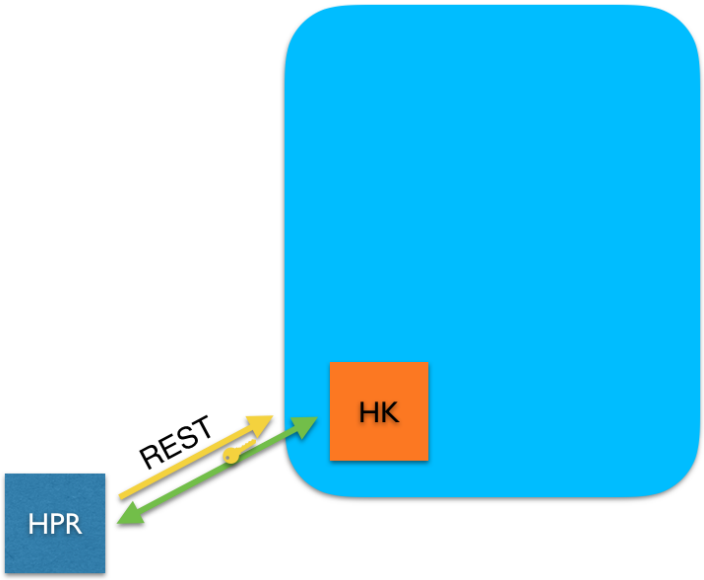

Fig. 2–8: Housekeeping Processor.

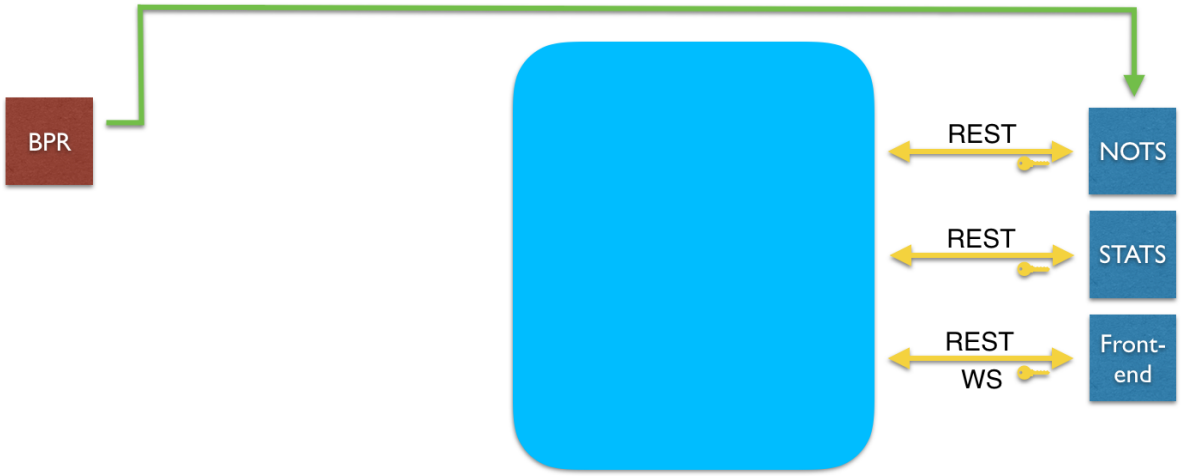

Fig. 2–9: Clients using the REST Interface.

Figure 2–9 shows all clients that use the provided REST interface.

- The Notification Service (NOTS) queries the microservices for metadata about every sensor on the satellite to determine e.g. maximum temperatures or other conditions that we want notifications for. After this metadata is obtained the Notification Service listens for all beacons parsed by the BPR. This service can now notify us about any values that are not as expected (not nominal) and, for example, post messages in a configured Slack channel.

- The Statistics Service (STATS) queries different Application Programming Interfaces (APIs) to provide us with daily statistics about the usage of our Operations and Satellite Systems. This includes e.g. the number of user logins for the OPS web page, the number of executed commands, successful commands, and executed and successful file transfers.
- The Frontend represents the OPS web page that developers as well as operators use to visualize all available data received from the satellite and command the satellite. It uses the HTTPS REST interface as well as websockets to get data from the microservices. The use of websockets (permanent connections of the client to the OPS servers) allow to automatically receive new data from the satellite in the users browser window once it is available. More information about this topic can be found in the IDP report of Thomas Zwickl.

3 Semi-Automated Bug-Tracking with Elfriede

Elfriede is a Slack bot (an application connecting to the messaging platform Slack) that automates certain processes. The following paragraphs define the problems that Elfriede solves, and explain how these problems have been addressed.

3.1 Problem Statement

It is well known that communication effort grows more and more the higher the number of participants gets. Our MOVE-II CubeSat project reached a size of about 100 active members at the beginning of 2017.

We used Slack as our main communication platform for the following purposes:

- Announcements to the whole team
- Coordination between sub-teams
- Coordination of the sub-teams
- Coordination of scarce resources (hardware, tools, rooms)
- Troubleshooting of problems
- Questions and Answers from and by anyone

Slack is an application that can be used via the browser or a separate application available on the Slack website or the different application stores (e.g. Google Play Store). It allows communication in different channels (chat rooms) and private messages. Messages can be used as a topic for a longer discussion associated to the original message (thread). For further information please visit the official Website: What is Slack?[1]

In addition to Slack the project management software Redmine was used for issue tracking, e.g. tracking and documenting bugs in hardware or software, missing documentation or features that still have to be implemented.

Most of the time issues and questions are discussed on Slack. Here team members can be directly addressed and asked for information, e.g. if something is expected behavior, how some task can be accomplished or how the current behavior of hardware or software can be explained. These questions often originate from missing documentation or faults in hardware or software.

The follow-up and mitigation or documentation of these faults is crucial for the success of our project. Everybody was therefore asked to create new issues in our issue tracker whenever something worth tracking is observed.

Due to the missing integration between Slack and Redmine this always was not effortless:

[1] `https://get.slack.help/hc/en-us/articles/115004071768-What-is-Slack-`

- The Slack environment has to be left (new browser tab or new application window) and therefore the context has to be changed
- User credentials have to be provided for login
- The issue has to be created, including
  - Title
  - Description
  - Assignee
- The link to the created issue has to be posted in Slack to enable others to follow-up on the issue

This led to the problem that many issues were never tracked and therefore forgotten or not followed up. While assigning people with the dedicated task of finding and documenting these issues improved the situation partially it did not lead to the expected results and there were still discussions, problems and solutions that never were tracked and documented.

3.2 Solution

To mitigate the aforementioned problems regarding the different used tools some solutions to bridge this gap were evaluated. This included the different integrations provided by the Slack platform for Trello and Redmine, the integrations provided by the Trello platform and the well-known platform Zapier. All these tools did not provide all the necessary functionality that would allow to automate our custom process.

Therefore a custom solution needed to be implemented. The goal of this solution was to seamlessly integrate our issue tracker into our Slack communication.

For this purpose a bug tracking workflow was introduced: Whenever something worth tracking is written a “bug” emoji shall be added as a reaction to this message. As soon as this message was then automatically transferred into our issue tracker a “checkmark” emoji is added as a reaction to this message to signalize that this issue was permanently and successfully tracked, or a “no entry sign” if an error occurred.

This workflow is visualized in Figure 3–1.

This was then implemented as an application that observes our Slack for new bug reactions and creates issues for them accordingly.

3.3 Implementation

The Slack bot was implemented using the Python programming language. This was due to the good availability and documentation of already existing packages that allow receiving notifications about new reactions and messages and writing messages on Slack.

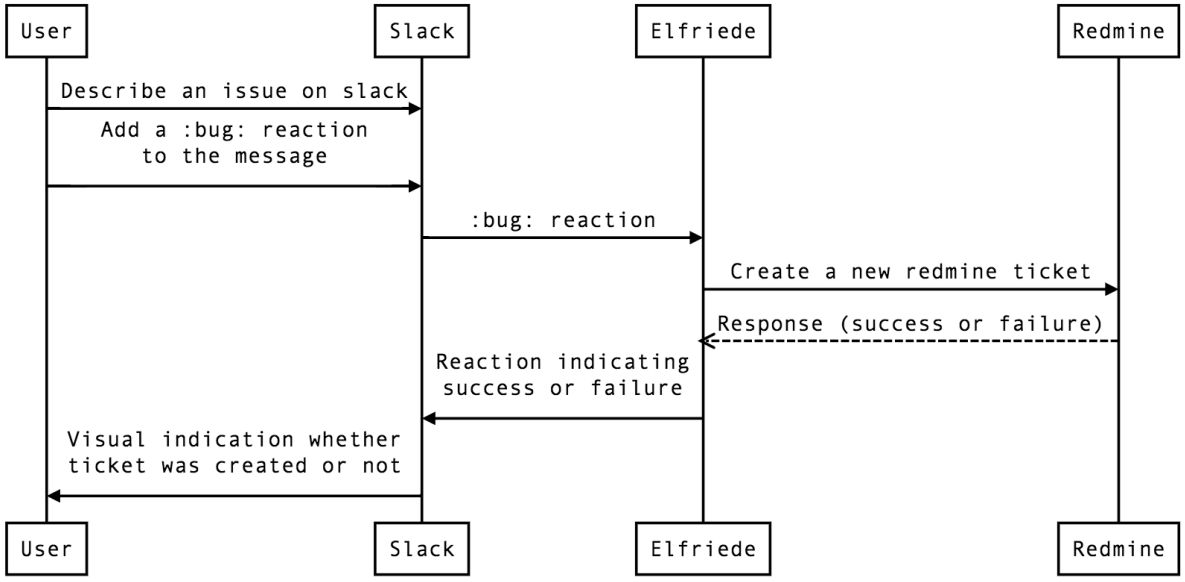

Fig. 3–1: Task Handler Workflow.

3.3.1 General Concept

The application is able to receive any events defined in the Events API[2].

For these events handlers can be programmed and then assigned to certain events via a configuration file. These event handlers execute arbitrary code for every received event and give a high flexibility to extend the existing application with additional functionality.

Currently the following handlers are implemented:

- Task Handlers
- File Handler

3.3.2 Task Handlers

The task handlers handle all functionality regarding tasks (called “Issues” in Redmine). Currently there are 2 task handlers implemented, one for Redmine issues, and one for Trello cards. The functionality of all task handlers can be split into 2 parts:

- When certain emojis are added as a reaction to a message, the task handler creates a new task for the associated emoji on the configured platform (e.g. Redmine). This is currently implemented for Redmine and the team organization tool “Trello”.
- When a new message is written in the thread of a message tagged as bug, this message is added as a new comment in the issue tracker. This documents the

[2]https://api.slack.com/events-api

whole discussion for an issue on Slack and allows to see the whole picture in the issue tracker, without having to switch back to Slack.

How this workflow looks like can be seen in Figure 3–2. By marking this message with the bug reaction “Elfriede” was triggered and created a Redmine ticket with its exact message as content. Every further message posted below Elfriede’s will be automatically added as a new comment in the Redmine ticket.

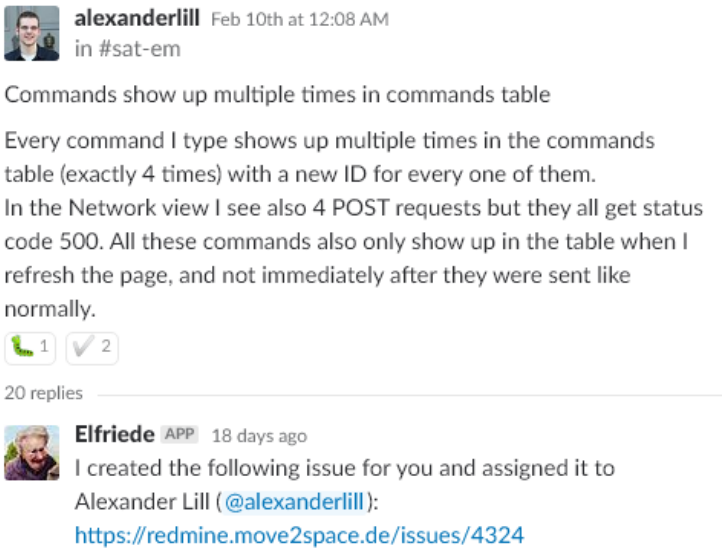

Fig. 3–2: Task Handler Example.

3.3.3 File Handler

The file handler solves an issue that we observed when commenting on uploaded files: The comments are posted in all channels where the file was posted. This often reduces readability and overview in a channel, as the discussion in the channel might have continued, while comments for files that were uploaded in the past always show up as new messages, and therefore destroying semantic locality in channels.

To solve this issue the file handler posts a message below every file that can then be used as a thread only for this file. Therefore commenting on files is not necessary any more.

Figure 3–3 shows an uploaded picture and the thread created by “Elfriede”. It is also visible that the created thread already contains 5 replies, that would otherwise clutter the overview of the channel.

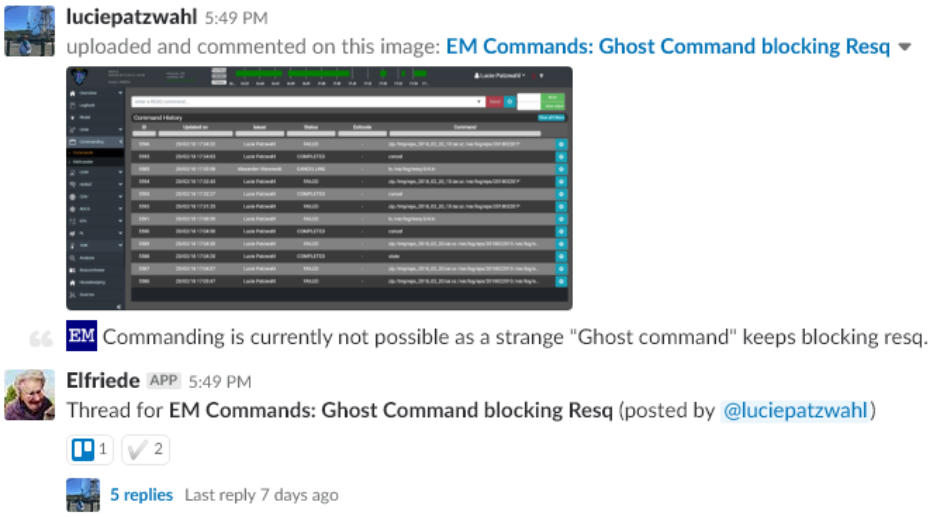

Fig. 3–3: File Handler Example.

3.4 Results

So far Elfriede has created 130 Redmine Issues and countless threads, and no manual observation of Slack channels as well as manual creation of Redmine issues is necessary any more.

4 Conclusion

This Interdisciplinary Project at the Chair of Astronautics can be summarized as the most interesting time of my Bachelor's and Master's studies. Reflecting on the amount of new knowledge and the many chances to put learned theory into practice I never learned as much in 2 years as I did now.

I am glad that I joined the project in May 2016, even though many, many hours have been spent on it I would not know how these could have been spent better than in this huge, interesting and interdisciplinary project.

Here I would like to thank many different people for their cooperation, enthusiasm and countless explanations of things that I did not know or still have only a very limited understanding of.

I especially want to thank Sebastian Rückerl for welcoming me into this team and Florian Schummer for his support in my very first weeks as Software Engineer in this project and ever since. Furthermore, I want to thank Martin Langer for his continuous motivation for this project and all the help that he provides everyone involved. Finally, I want to express my gratitude for all the people I was able to get to know and work with. The things we accomplished would not have been possible without the Operations team and the support of the rest of the MOVE team.