SpaceOps-2021,12,x1480

A New Operational Concept for Stratospheric Science Payloads: Reusing an Extensible Satellite Framework for Operating Regular Balloon-Based Astronomical Missions

Mahsa Taheran Vernoosfaderani[a]*, Philipp Maier[b], Sara Bougueroua[c], Andreas Pahler[d]

[a] *Institute of Space Systems, University of Stuttgart, 200 Chunghwa Street, Pfaffenwaldring 29, D-70569, Stuttgart, Germany*, mtaheran@irs.uni-stutgart.de
[b] *Institute of Space Systems, University of Stuttgart, 200 Chunghwa Street, Pfaffenwaldring 29, D-70569, Stuttgart, Germany*, mtaheran@irs.uni-stutgart.de
[b] *Institute of Space Systems, University of Stuttgart, 200 Chunghwa Street, Pfaffenwaldring 29, D-70569, Stuttgart, Germany*, mtaheran@irs.uni-stutgart.de
[b] *Institute of Space Systems, University of Stuttgart, 200 Chunghwa Street, Pfaffenwaldring 29, D-70569, Stuttgart, Germany*, mtaheran@irs.uni-stutgart.de

* Corresponding Author

Abstract

Stratospheric balloon observations fill the gap between expensive space telescopes and ground based observatories, which have limited functionalities in the ultraviolet and infrared spectrum. Even though stratospheric observations have a long history, they have been mostly one-shot missions. The European Stratospheric Balloon Observatory *Design Study* (ESBO *DS*), funded by the EU Horizon 2020 program, will develop an infrastructure for regular astronomical observations from the stratosphere, creating a new operational concept for stratospheric science missions. ESBO aims at complementing space-based and airborne capabilities over the next 10-15 years and at adding to the current landscape of scientific ballooning activities by performing regular flights and offering an operations concept that provides researchers with a similar proposal-based access to observation time as practiced on ground-based observatories.

To reach a balloon-based service-cantered infrastructure for astrophysical observations, we need to accommodate different instruments in a short time-span and efficiently use the flight time. Therefore, a major element of this infrastructure is an extensible and reusable on-board software for remote operations and automatic control. The software design has several features to strengthen the reusability and extensibility. These include : a reusable core software framework; component templates to facilitate implementing flight software for scientific instruments; independency of data interfaces from scientific devices; hierarchical structure of software components to easily add automatic operations and control system behaviour; and last but not least, standardized and extensible balloon-ground interface by using the Packet Utilization Standards (PUS). The core software framework is independent of platform and operating system. It is based on a component-based satellite software framework developed for the Flying Laptop (FLP) satellite, adjusted for specific requirements of stratospheric missions. The common interfaces defined within the software facilitate adding device specific software components to the core, which handles communication between components and manages on-board processes.

This paper describes the architecture of this software and the development process undertaken for the first ESBO prototype mission, STUDIO, to be launched in 2022. The main features of the on-board software are presented in detail, showing how they contribute to an extensible software and create potential for automatic operations in future. The status of software implementation, integration and tests are discussed. Furthermore, the challenges of such an architecture for integration and tests, in particular in a mission with several Commercial of the Shelf (COTS) instruments and a distributed team of developers, and the lessons learned in the past two years of development would be presented in detail.

Keywords: Scientific Ballooning, Flight Software, Software Framework, Reusability, Extendibility

Acronyms/Abbreviations

Astrophysics Stratospheric Telescope for High Spectral Resolution Observations at Submillimetre- wavelengths (ASTHROS)
Attitude Control Unit (ACU)
Application Programming Interface (API)
Balloon-borne Imaging Telescope (SuperBIT)
Balloon-borne Large Aperture Submillimetre Telescope (BLAST)
Consultative Committee for Space Data Systems (CCSDS)
European Stratospheric Balloon Observatory (ESBO)
ESBO Design Study (ESBO DS)
Flight Software Framework (FSFW)
Flying laptop (FLP)
Far Infrared (FIR)
Front End Equipment (FEE)
Global Positioning System (GPS)
Image Stabilization System (ISS)
Institut für Raumfahrtsysteme (IRS)
Microchannel Plate (MCP)
innOvative Research Infrastructure based on Stratospheric balloONs (ORISON)
Packet Utilization Standards (PUS)
Software Development Kit (SDK)
Stratospheric UV Demonstrator of an Imaging Observatory (STUDIO)
Swedish Space Corporation (SSC)
Telecommand (TC)
Telemetry (TM)
Telescope Instrumentation Platform (TIP)
Ultraviolet (UV)
Visible (VIS)

1. Introduction

The idea of using stratospheric balloons to overcome the opacity of Earth's atmosphere for astronomical observations is not new. Historically, the advantages were obvious: spacecraft did not exist or were hardly available and capabilities of airplanes were limited, leaving balloons as the only option to move instruments above most of the atmosphere. Nowadays, the benefits do not seem as clear: both spacecraft and airplanes provide powerful observation platforms and ground-based telescopes invest large efforts into compensating atmospheric influences. However, even in the era of nano and microsatellites, space observatories are intrinsically expensive and bear operational limitations: development times are long, updates or corrections of the instrumentation are usually not possible after launch, and consumables, such as cryogenic coolant fluids, cannot be refilled or replaced, as seen with the Herschel Space Observatory. Furthermore, rather conservative approaches towards new technologies are used to minimize risks of expensive failure. Ground-based and airborne telescopes, on the other hand, still suffer from fundamental limitations imposed by the atmosphere at certain wavelengths.

On the other hand, technological advances have made balloon-borne telescopes more attractive over the last couple of years. These particularly include more reliable balloons and the opening of long-duration flight routes, allowing flight durations of 30 to 40 days on "conventional" long duration routes and promising 50 to 100 days on ultra long duration routes. Consequently, the last couple of years saw an increase of balloon-astronomy initiatives aiming at more regularly flying missions rather than the more common "experiment"-type of flights. Noteworthy recent examples are the U.S./Canadian Super pressure Balloon-borne Imaging Telescope (SuperBIT) [1], the JPL-lead Astrophysics Stratospheric Telescope for High Spectral Resolution Observations at Submillimetre-wavelengths (ASTHROS) [2], and the U.S.-lead Balloon-borne Large Aperture Submillimetre Telescope (BLAST) [3] along with its successors.

The goal of ESBO is to further lower the entry barrier to balloon-based observations by providing an operating institution that offers observing time and instrument space on balloon-based telescopes. Toward this

goal, a reusable and extendible flight software, the design of which is presented here in this paper, is under development for our first flight in 2022.

The following section presents an overview of the ESBO concept and the design of STUDIO, the UV/visible prototype of ESBO. In section 3, the architecture of ESBO flight software is described, emphasizing the core framework and its features. Section 4 describes the drawbacks and challenges of this architecture, which we have experienced over the past year during development and tests.

2. The ESBO concept

Balloons can cater a partly overlaying, partly different parameter space of mission requirements compared to space missions. On the one hand, individual flights are considerably shorter than space missions. Depending on the exact wavelength, some atmospheric absorption may still exist. Particularly interesting for infrared observations, cooling of large structures, e.g. telescope mirrors, to cryogenic temperatures is much more difficult than in space.

On the other hand, the comparatively much lower mission costs in combination with the shorter flight durations (and re-flights to obtain more observation time) allow a less risk-adverse approach. This, in turn, allows shorter development times, connected with the possibility to fly more up-to-date instrumentation. Regular returns of instruments furthermore allow their update and re-arrangement, including, of particular interest for infrared observations, the possibility to refill cryogens. Balloons furthermore come with less solid restrictions to the geometrical size of payloads, which are basically suspended in free air underneath the balloon.

These aspects suggest to use balloon-based observatories not as much to obtain extremely high sensitivity, but rather to obtain high spatial resolution, to carry out surveys, to cater rapidly developing or versatile instruments, and to address instrumental and observational gaps that for one reason or the other are not covered by space-based observatories (such as heterodyne instrumentation for the FIR, for example). Given the lower required investments and smaller timescales, they also tend to allow more flexibility and adjustability for specific scientific interest

The ongoing ESBO Design Study (ESBO DS) builds on the results of the ORISON project [4], and therefore represents the second step towards ESBO. Under ESBO DS, the full infrastructure, with particular technical focus on FIR observational capabilities, is being conceptually designed. The long-term goal for ESBO is to establish an operating organization that conducts regular flights (1+ per year) of balloon-borne telescopes. At least the gondola and the respective telescope(s) are thereby foreseen to be provided by the operating organization as well. With regard to instrument provision, we regard it as desirable to foresee both "facility instruments", with open access to observation time, as well as flight opportunities for "PI instruments". This approach should make ESBO a service provider for both instrument builders and observers.

These considerations require that the ESBO infrastructure does not only include flight systems, but that it also offers tools and procedures for the instrument preparation, proposal, observations, and data reduction phases

While the current concept foresees that an ESBO institution provides flights of balloon-borne platforms, it also foresees that the operation of launches and of the balloon itself during flight be procured from currently existing and experienced launch and flight providers. More details on the envisioned ESBO infrastructure and its scope can be found in Maier et al. (2020) [5]

2.1 STUDIO – The UV / visible Prototype Mission

STUDIO is currently under production and integration and will undergo further integration and test activities in 2021. Mating of the payload, primarily the telescope, and the gondola is foreseen for spring 2021, with subsequent full-system tests on ground. The first flight of STUDIO, is currently planned for the autumn turnaround conditions flight window over Kiruna, Sweden, in 2022.

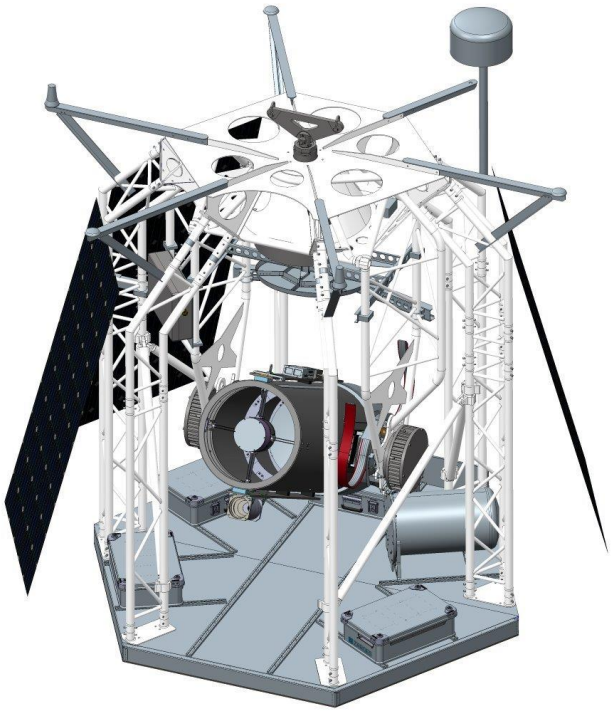

Fig. 1. STUDIO gondola. One solar panel is not shown for better visibility of the payload. Most electronics and service systems are located on the gondola floor.

The gondola / bus supports the payload with all essential service systems. The mechanical structure is designed from COTS aluminum tube structures in combination with custom-made tubing structures and holds the payload and all the subsystems. The idea is to have a flexible design that can be easily modified for different payload requirements and sizes, and that can also conveniently be disassembled in the field for fast and easy recovery. The telescope and the pointing system are mounted directly in a stiff inner gimbal-like structure, as it can be seen in Fig. 1.This also makes it possible to test the pointing system with the telescope and the inner structure without the rest of the gondola. When the telescope is placed in its parking position it is covered by a lid that protects it from dust. Most of the subsystems are placed in boxes on the floor of the gondola to achieve a low center of mass for the gondola. This decreases the risk of tipping the gondola at landing and avoiding damages to the telescope and instrument.

The pointing system is a further development of the pointing system [6] developed for the astronomical PoGO balloon missions. It is equipped with an elevation motor directly operating on the telescope and azimuth motors and a fly wheel for turning the complete gondola including the telescope. The pointing system uses multiple sensors, but the most essential ones are three differential GPS antennas placed on booms at the top of the gondola and a star tracker manufactured by Arcsec placed on the telescope. Further sensors can be added to the system depending on mission requirements and the system can handle heavier telescopes / instruments up to 800 kg.

STUDIO's optical payload is composed of a 50 cm aperture closed-tube telescope, with the Telescope Instruments Bench (TIP) attached to the back of the telescope. Fig. 2 shows the Telescope and TIP assembly, along with the interfaces to ACU's gimbal and star tracker. The telescope is a modified Dall Kirkham in an f/13 configuration. It features a 50 cm elliptical primary mirror, a spherical secondary mirror and three lenses for field correction located in the centre hole of the primary mirror. The secondary mirror is moveable via three remotely operable actuators. These are used to move the telescope's M2 mirror with a resolution of ~3 µm, while the telescope's depth of focus is ±30 µm at 180 nm.

The TIP includes two principal instruments:

- An advanced photon-counting, imaging microchannel plate (MCP) detector that performs observations in the UV band (180 nm – 330 nm), described in section **Error! Reference source not found.**
- A commercial visible light camera PCO.edge 4.2, used mainly as the tracking sensor in a closed-loop fine image stabilization system, which will also serve as a secondary scientific instrument to cover observations in the VIS band (350 nm – 1000 nm)

Additionally, the TIP houses a fast steering mirror mounted on a commercial Tip / Tilt Platform as part of the image stabilization system to achieve 1 arcsec pointing stability.

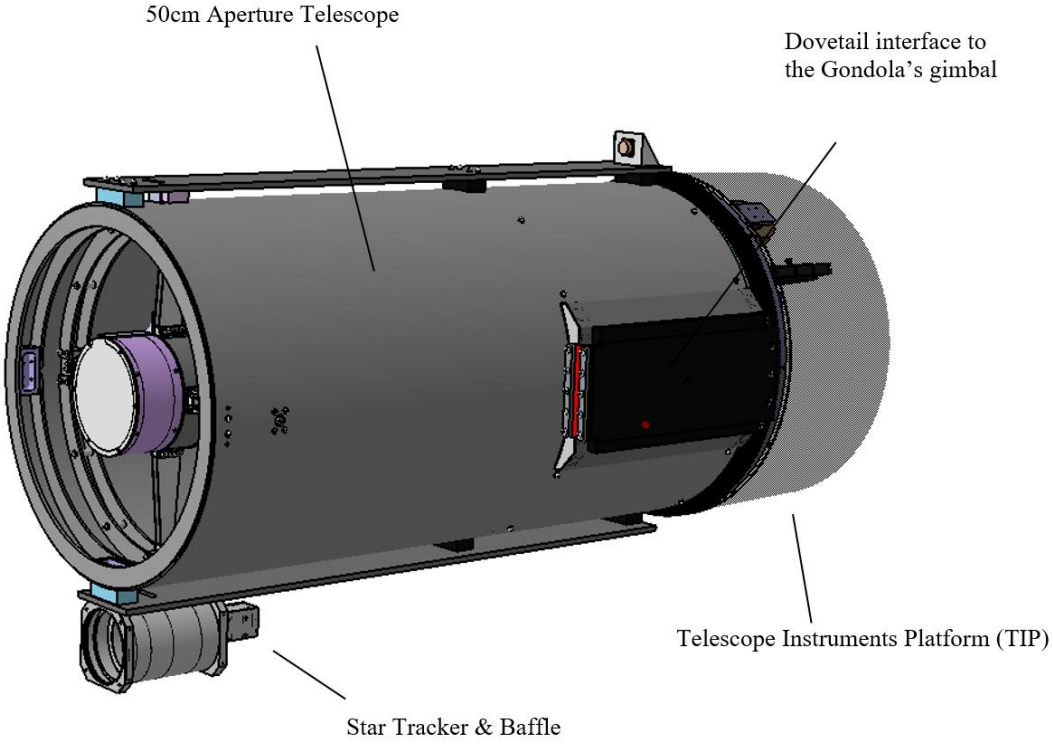

Fig. 2. STUDIO telescope and TIP assembly

For more details on the design of the STUDIO, please refer to Maier et al [7].

3. ESBO flight software architecture

3.1 Flexibility in STUDIO on-board software

As mentioned in the introductory section, the long-term goal of ESBO is to become a regular-flying service provider, providing different telescopes, facility instruments, and opportunity to accommodate user-provided instruments.

Keeping in mind this objective, a driving requirement for the on-board software of ESBO is to be able to accommodate different instruments for different flights. Given that the vision is to fly regularly, integration of the mission-specific instruments in the software should be facilitated. There are two general approaches for such a system:

- Distributed operations, in which instruments would be provided with specific power and data bandwidth and operates independently.
- Centralized operations controlled by a core on-board software.

The first approach would not optimally use the observation time and the data bandwidth available. While using a centralized on-board software requires integration of different instruments with the software, costing us time and resources, it adds a lot of potential for scientific operations, allowing us to define observation scenarios with a subset of instruments and optimally planning the observation sequence, particularly if a failure in one instrument happens. It would also be easier to modify all parts of the software, including instrument components. Therefore, we decided to use an in-house developed core framework for small satellite missions with the following major advantages:

- Possibility to reuse most of the existing software for each mission, changing the instruments, the telescope, or changing the service systems, would not require any changes in the core functionalities.

- Facilitating integration of new software components, aka new instruments, for future missions, and the possibility of reusing the already implemented interfaces for new instruments.

- Having a platform-independent, and OS independent software

3.2 Flight Software Framework (FSFW)

The on-board software for STUDIO is based on the Flight Software Framework (FSFW) that was developed for the Flying Laptop satellite and has a proven history of flight since 2018. The first version of the framework, developed at the Institute of Space Systems (IRS) is licensed open-source, and is currently used in other small satellite missions of the institute as well. For an in-detail paper on the FSFW please refer to to Baetz et al. [8].

FSFW is a component based object-oriented framework developed in c++, which implements common functionalities of the space flight software. It is a platform independent and operating system independent software, including abstraction layers for Linux, RTEMS, and Windows. Some of the components of the core framework are shown in Fig. 3. Mission specific components connect to the FSFW by implementing specific interfaces.

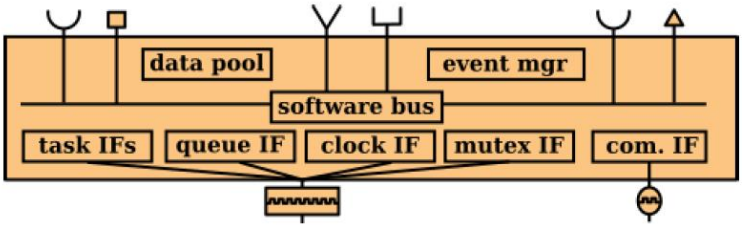

Fig. 3. Major components of the FSFW [9]

3.3 Templates/Base classes

FSFW provides Base classes for different types of mission-specific components. These base classes can be inherited by subclasses to implement new mission-specific software components, by providing implementations for adaptation points to define component-specific behaviour.

Any mission specific component is an instantiation of a C++ class, which must implement certain interfaces to allow interaction with other components and the FSFW. It is possible to bypass these base classes and connect to the core framework by implementing the common interfaces. Nevertheless, these templates are generally a good starting point for most of the devices. All these components are reusable, and can be used in other mission without changing the source code.

The following templates are included in the FSFW:

- **Device handler** components control and monitor the equipment, like detectors and different sensors and actuators. They include and encapsulate all device-specific properties, communication protocols and device modes and states. Device handlers can also determine if a device is working normally, is healthy, or not. Assemblies group redundant devices together, and are responsible to switch between devices if one is not healthy.
- **Controller** components perform some form of control activity : like thermal controller, pointing controller, and similar
- **Subsystem** components are composed of a set of devices, assemblies and controllers which together perform a specific functionality and therefore their state should be coordinated
- **Ground service** components provide specific functionality relevant for ground interaction.

It should be noted that these components all exchange information with each other through the common core software bus or datapool, and not directly.

3.4 Hierarchical system definition and configuration

The core software framework provides flexible mode-based operations on-board. Automatic operations can be carried out in different operational modes, just by setting the mode of a component. Mode commands can be applied to devices, controllers, subsystems and the whole system. Change of a mode in a higher-level component automatically changes the modes of lower level components. Devices and controllers can be grouped together in subsystems, and if one fails to operate normally, the whole subsystem configuration can automatically respond. The modes can be controlled from ground.

While STUDIO does not include redundancies for most of the devices in its first prototype mission, this feature will be used significantly in future missions to add more automation and increase reliability and robustness. Fig. 4 shows three of our subsystems that are composed of more than one component. Visible instrument handler would be both part of the visible imaging subsystem, paired with its filterwheel, and the image stabilization system, paired with ISS controller and Tip/Tilt device handler. This is because visible instrument is used as a sensor of the stabilization system during observations with UV. Nevertheless, it is foreseen that we might successfully operate it during the day for visible science cases as well.

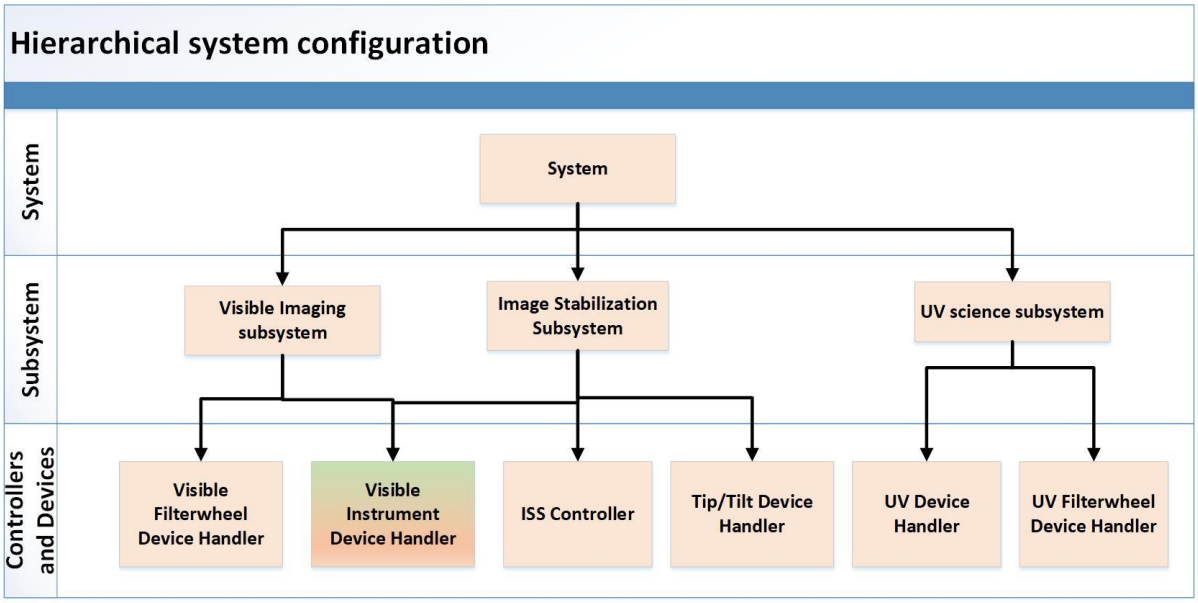

Fig. 4. An example of hierarchical system configuration for STUDIO

3.5 Packet Utilization Standards

While the FSFW does not limit the space link protocol, it uses CCSDS space packets for the application layer. All the functionalities to encode and decode the space packets are included within the framework. By using the Flight Software Framework, We are right now constrained in our format of communications to and from ground. In addition, FSFW is implemented for missions using Packet Utilization Standards.

FSFW has two base classes for implementation of PUS services, depending on the functionalities required for the specific services. In Fig. 5 these two service classes and their connection to other components are shown. The so-called gateway services convert the ground message to the internal messaging protocols understandable by devices and controllers. Standalone services do not need to forward the message received to other components on board, and therefore have a different implementation. For example, scheduling service is responsible to create and maintain a time-based schedule, and release a command to other service gateways at the right time. Therefore, it does not forward the messages to any device or controller, and is a standalone service. On the other hand, functional commanding using service 9 of PUS is sent to a service gateway, where it is translated to an action message, and sent to a specific object, which has implemented a proper interface, to be able to perform an action upon request.

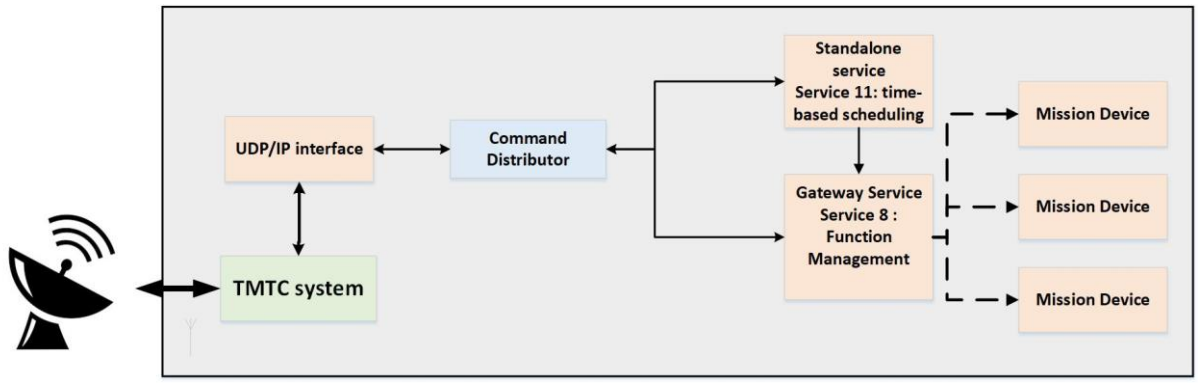

Fig. 5. Architecture of ground services, command distributor is a FSFW component, other software components are mission specific

Part of the PUS services that are commonly used by missions are now included in the core framework implementation. However, this is not a full implementation of PUS services and subservices. The user is able to use the base classes to implement new services required for a specific mission.

While there is no explicit dependency in Device Handler components on PUS services, the type of messages defined follows the standard. The potential of device handler base classes is therefore fully exploited, when used with PUS services. The devices expect different type of commands from ground, shown in Fig. 6, which are action messages (PUS service 9), parameter messages (PUS service 20), health and mode messages, the latter ones are custom PUS services not included in the PUS standard.

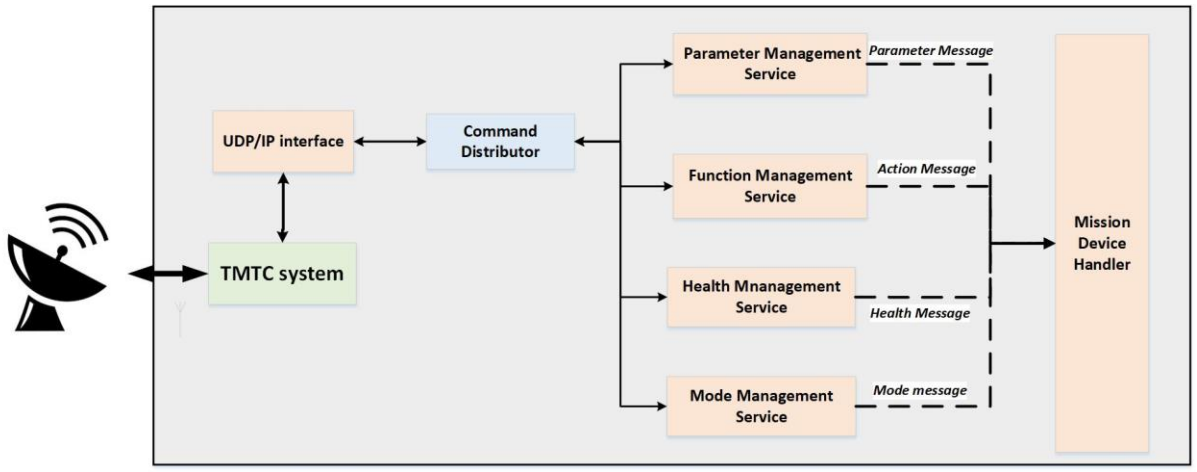

Fig. 6. Ground service commands to device commands conversion

Therefore, even though it is possible to extend the protocols supported, or bypass the core component related to CCSDS and PUS to use own-developed parts, we decided to keep this architecture and use PUS formatted messages.

3.6 STUDIO payload flight software architecture

Fig. 7 shows a breakdown of the STUDIO system. On board components are divided between gondola and payload. Payload computer uses the gondola communication system to communicate with ground, and to the pointing system. It also receives GPS data (position and time) and status of the communication link through this Ethernet interface.

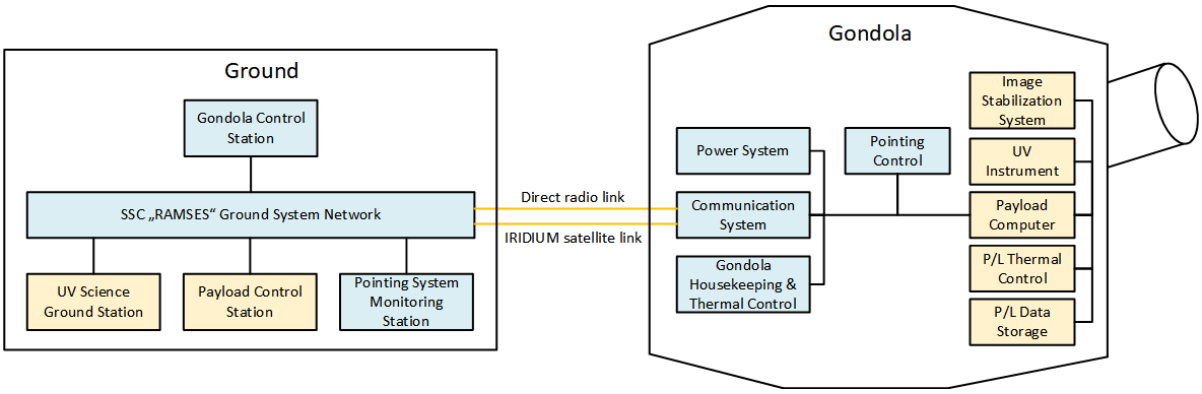

Fig. 7. STUDIO system architecture. Bus (gondola) components in blue, payload components in yellow

Payload flight software uses Linux operating system on an industrial single board computer. There are no hard real-time requirements for this mission.

Fig. 8 shows a high-level breakdown of the software and its components. In the figure, the division of FSFW and mission code is color-coded. Mission specific code includes several devices and controllers, all exchanging data with the global datapool and command distributor. On the other side, device handlers are flexible, as they are not implemented for a specific lower level interface. They can communicate with any communication interface as long as it is wrapped in the required software interface. A subgroup of PUS services have recently been added to the core framework to increase reusability of the framework for future missions. More PUS services are included in STUDIO mission code to complement the core services. Space packets are sent and received on a UDP/IP interface to the gondola TMTC system.

Given the limited bandwidth of the communication link to the ground, a traffic shaper is implemented that prioritizes and optimizes the actual data rate of different packet types in an adaptive algorithm for both science and housekeeping data.

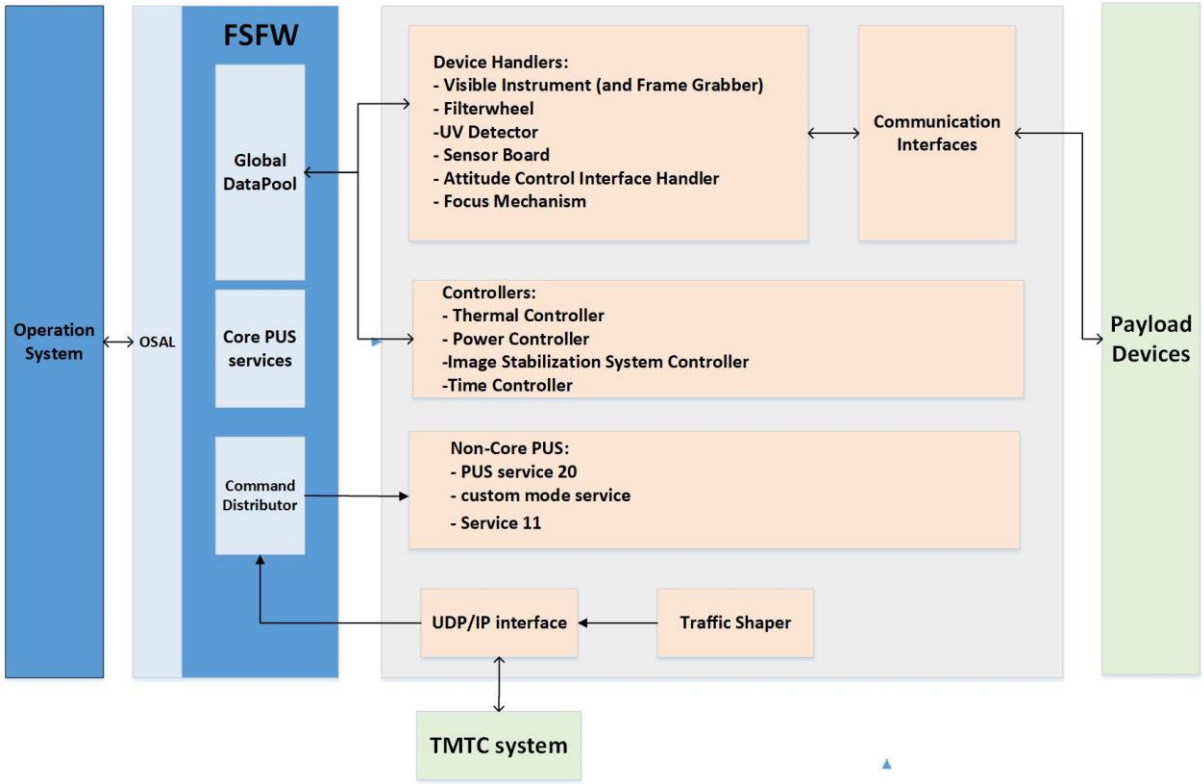

Fig. 8. Breakdown of ESBO flight software components

3.7 Operations of the STUDIO telescope

FSFW provides operators with several features that facilitate operations, particularly when it comes to automatizing operations. These features include:

- The concept of modes and mode management possibilities
- Inclusion of failure detection , isolation and recovery

This can be explained with an example from our telescope stabilization system. The stabilization system has the following modes:

- Off: all the components are off and the stabilization system is not active
- Standby: the controller is idle, and the actuator and the visible device are both in standby.
- Acquisition: In this mode, the controller is idle, the actuator is in standby mode and the visible device is in acquisition mode.
- Stabilization: In this mode, the controller is in stabilization mode with the actuator in active mode and the visible device is in stabilization mode.

Setting the mode of the stabilization subsystem automatically configures the three objects to switch to the defined mode.

If the visible device handler notices that the target is out of field of view through signal to noise ratio in a subframe, it issues an event. Similarly, if the controller detects that the centroid is out of the defined controllable region, it issues an event. These events are handled by a FDIR class in both cases, which can change the subsystem mode to a suitable one. In the former case, the stabilization mode needs to fall to Standby, and wait for the pointing to be fixed. In the latter case, the stabilization mode would fall to Acquisition, and waits for the acquisition to be validated.

The benefits of these mechanisms comes in to play for our scientific operations, when the system is not commandable from ground due to loss of communication. It also in high severity failures isolates the failures before sever consequences put the whole mission in danger.

3.8 Scheduling of the STUDIO telescope

Given that we use PUS services, telescope has an onboard time bases schedule, which can be modified both automatically and by the ground. The schedule can be loaded in to software upon initialization of the software, using an interface to a JSON file onboard. Each observation is a separate subschedule within the main schedule. They have the required commands for a full sequence of scientific observation, from pointing, acquiring the target, and the actual scientific operations of the detector.

The subschedules- observations- can be deactivated or activated again. Their starting time can also shift if needed. Individual commands within a subschedule can also be modified, and missing commands can be added.

Onboard events are used to automatically interrupt an observation in case of failures, and any observation that can not be performed in that state would be deactivated.

4. Drawbacks and challenges of using the FSFW

While using this architecture is advantageous for the long-term ESBO vision, it comes with its own drawbacks. The main reason is that FSFW is developed with requirements of small satellite missions in mind, and using it in a balloon mission requires some modifications, and implies some limitations. In this section, few of these disadvantages are described, with real-life examples from STUDIO mission software development.

4.1 Communication interfaces

A major difference of a balloon mission and a space mission is the approach to risk management and software reliability. In a balloon mission, given the lower level of environmental requirements and fewer restrictions on weight, it is possible to use a wider range of equipment on-board. Within STUDIO, we have been able to use many devices that are mostly developed for ground-based telescopes, by rigorous standard environmental testing. Most of these devices come with their own ready to use software, or at least software libraries or software development kits.

One specific example we faced at the beginning was with integrating the COTS devices that came with their own libraries to use. The framework encapsulates communication with the devices in a separate class called a communication interface. This separation is useful for two reasons:

- To change the actual interface of a device while reusing the device source code, e.g. changing the Can Bus to space wire, etc., without the need to modify the device component.

- It also lets the user to use the same communication interface for several devices.

However, the communication interface definition within the framework at this stage does not consider the use of third party SDKs, which is not a common approach for a space mission.

The easiest solution we used was to call library methods within the communication interface implementation for the device. It somehow breaks the encapsulation of device code and interface definition (meaning we do not benefit from the reusability of the interface anymore), but was from the perspective of time and cost constraints easier at this point.

However, in future, it might be useful to modify the framework to include another template for such devices, a base class that does not separate communication interface and the device specific behaviour implementation.

4.2 Distributed Computing

Another limitation of the current open source version of the FSFW is with distributed computing, as some of the core components work only in a single address space in current state.

This limitation as affected the implementation process of STUDIO on-board software, for the M2 focus mechanism of the telescope. M2 mechanism uses three step motors, which are controlled by a library developed in c# and incompatible with our c++ centralized software.

A first solution would have been to have a second machine running windows, but FSFW does not support distributed computing with different processors and one instance of FSFW. Adjusting the FSFW to have components of the same instance of FSFW be on separate processing units would create a lot of complexity.

Nevertheless, it is theoretically possible to have several instances of the FSFW on different machines, as seen in Fig. 9 on top, provided that a gateway component connects them together and forward inter-component communication to some on-board network, However, this requires an extension of the current version of freely available FSFW. This concept has been successfully implemented for the FLP2 platform at Airbus Defence and Space [9, 10].

A second solution, shown in Fig. 9 on the bottom, is to have a windows machine or virtual machine with the M2 control software implemented in a way that it communicates through an interface with a dedicated device handler on the main computer. While this is also theoretically possible, it would add a complex interface, and complicates the operations of the system, reducing the overall mission reliability.

We finally, given these limitations and the experience of two more teams using the same mechanism in past missions, decided to replace the whole controller of the mechanism, developing a full system to control the motors.

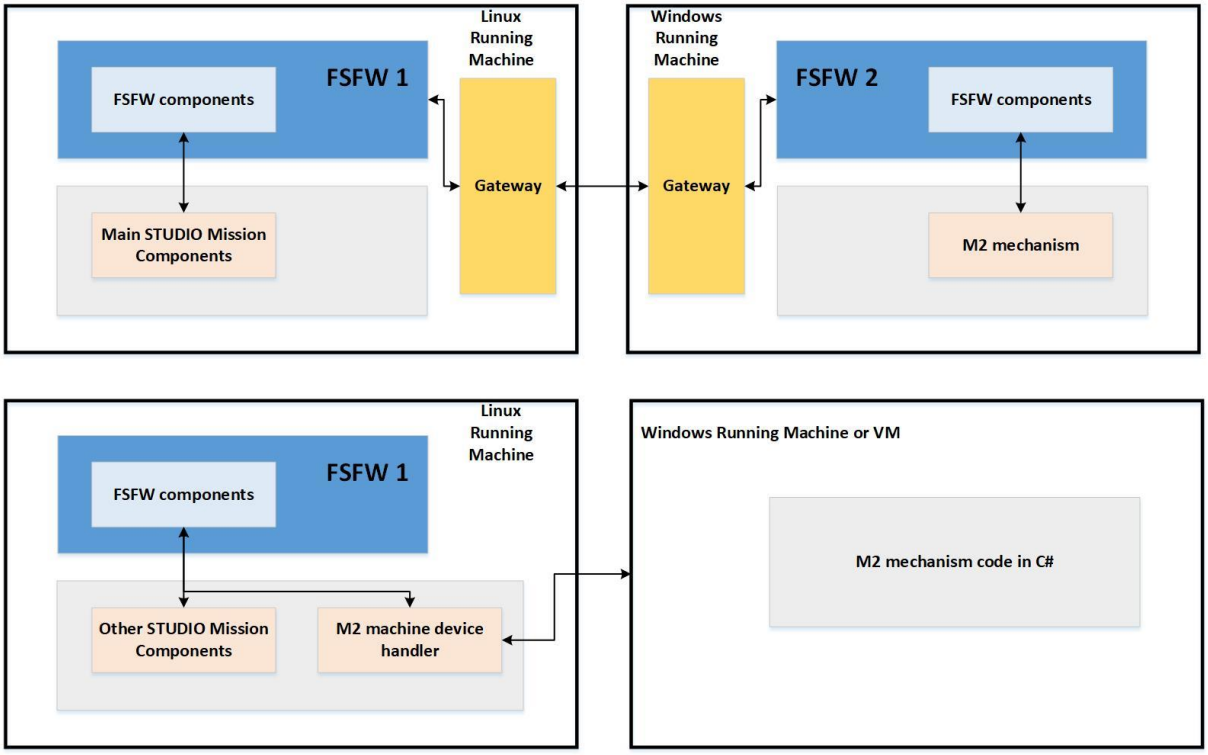

Fig. 9. Architectural solutions to M2 mechanism

4.3 Ground software

As mentioned above in section 3.5, the use of framework implicitly limits our choice of communication protocol to CCSDS space packets in application layer and PUS services. This limitation extends to the ground operations software as well. We therefore had to choose our ground operations software among the systems commonly used for space missions, which are notoriously complex and not suitable for smaller missions such as a short duration balloon flight, or in a different approach develop a lighter operations system for ESBO.

Given that the ground operations of the STUDIO Gondola is separately performed using RAMSES software suit, the simpler and cheaper solution was to use RAMSES as well for payload operations. RAMSES is considerably simpler than the other ground operations systems and used by Swedish Space Agency for balloon and sounding rocket missions.

Fig. 10 shows the main components of RAMSES ground operations.

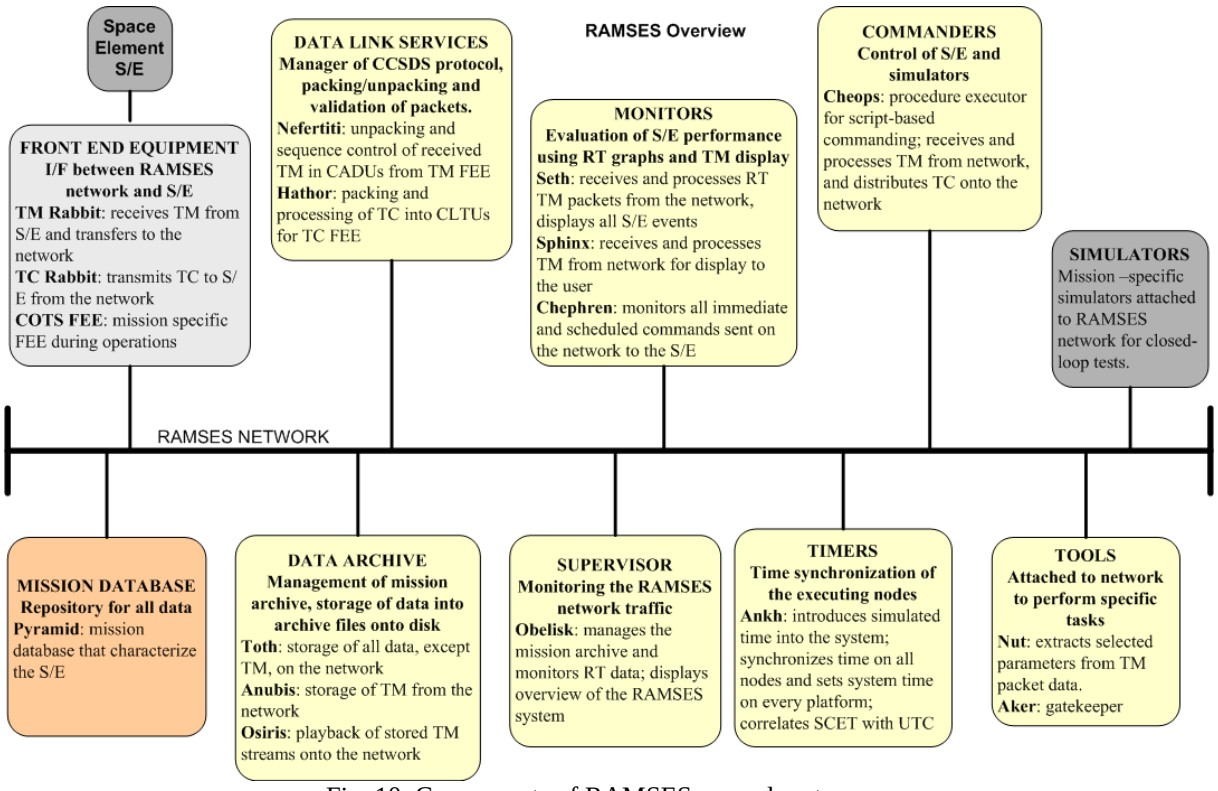

Fig. 10. Components of RAMSES ground system

On the other hand, it should be noted that for science data produced by our UV detector and our visible instrument, we would bypass the framework components to create space packets, and using a separate transparent UDP/IP connection provided by Gondola TMTC system, we would send the data using nftables in Linux. Fig. 11 shows a schematic of this architecture.

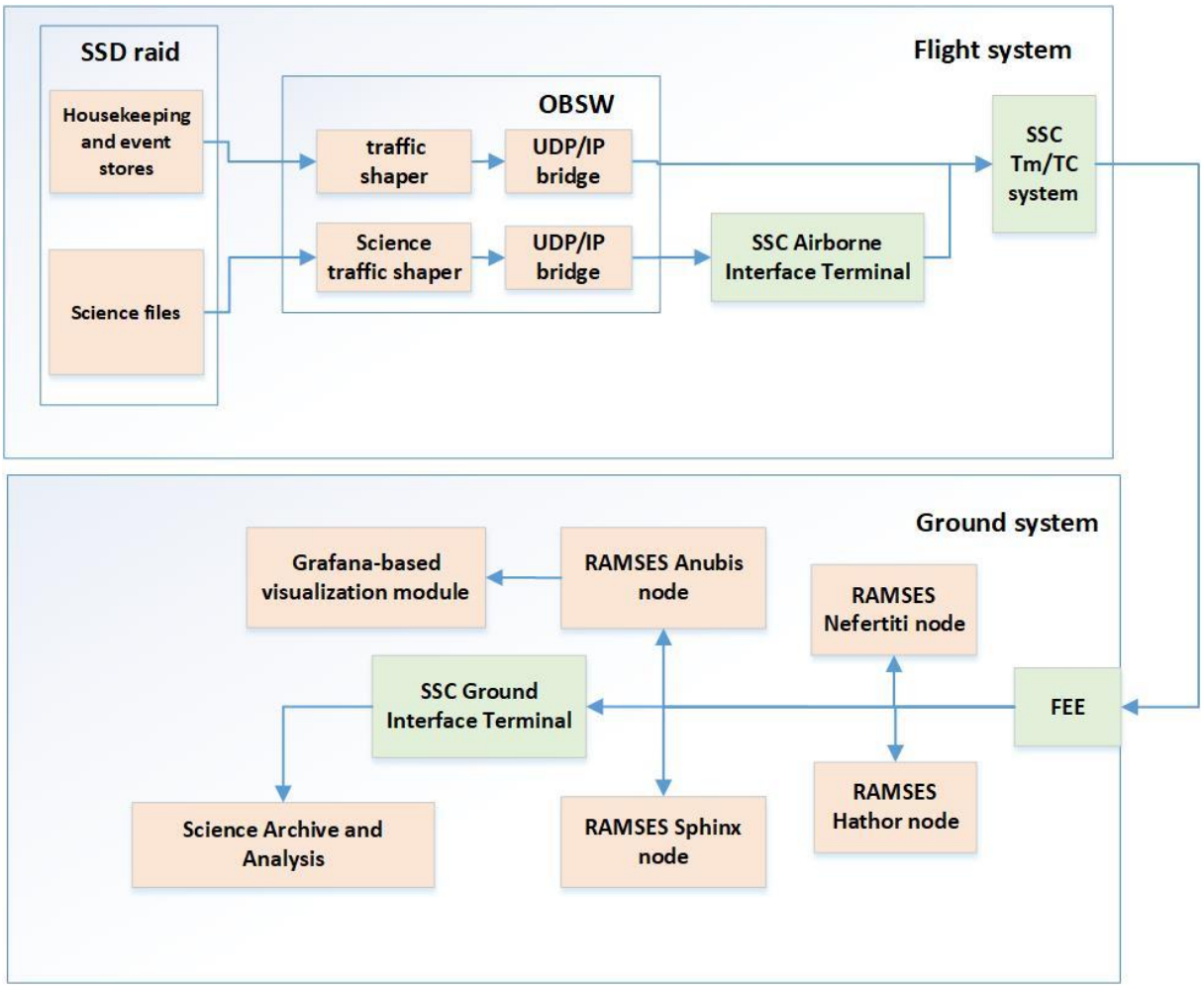

Fig. 11. Space-Ground link from payload on-board computer

4.4 Integrated and testing of FSFW based on-board software in a distributed development environment

One major challenge we faced within the process was implementation at subcontractors and scientific teams. Using a core framework, especially one that is recently developed in an academic environment and might not be well documented, requires some learning time, and getting acquainted to. FSFW is a white box framework, and working with it requires some knowledge of the internal processes.

We have proceeded the implementation with two external teams, one in Tubingen for STUDIO UV detector, and one in Spain for our Tip/Tilt platform and stabilization controller. To improve the learning curve for developers, some simple device handling examples have been developed over the last year to be a starting point.

What we have experienced in development with the framework is that for each component, we have started with developing simpler codes independent and testing independent, so it might add some time and resources to development. Given the interdependencies of the framework, testing a device code requires one of the following to be in place:

1- The ground mission operations system , able to send space packets and receive and decode them
2- A dedicated test software with the ability to send and receive space packets
3- A command injector component on the framework

Using a command injector is also a limited possibility. Given that any component integrated with the framework also cannot be commanded directly, a command injector can follow a readily planned sequence of commands. The first option is also not ideal in a distributed development environment, with several team working in different locations on different components. Therefore, a lighter test system would be a necessity for the test and integration process.

4.4.1 Integration and test in practice: the case of Image Stabilization System

One representative case of distributed computation architecture, development in various locations and testing is the Image Stabilization System for the STUDIO telescope. Fig. 12 shows the breakdown of the Stabilization system and the data flow between these components.

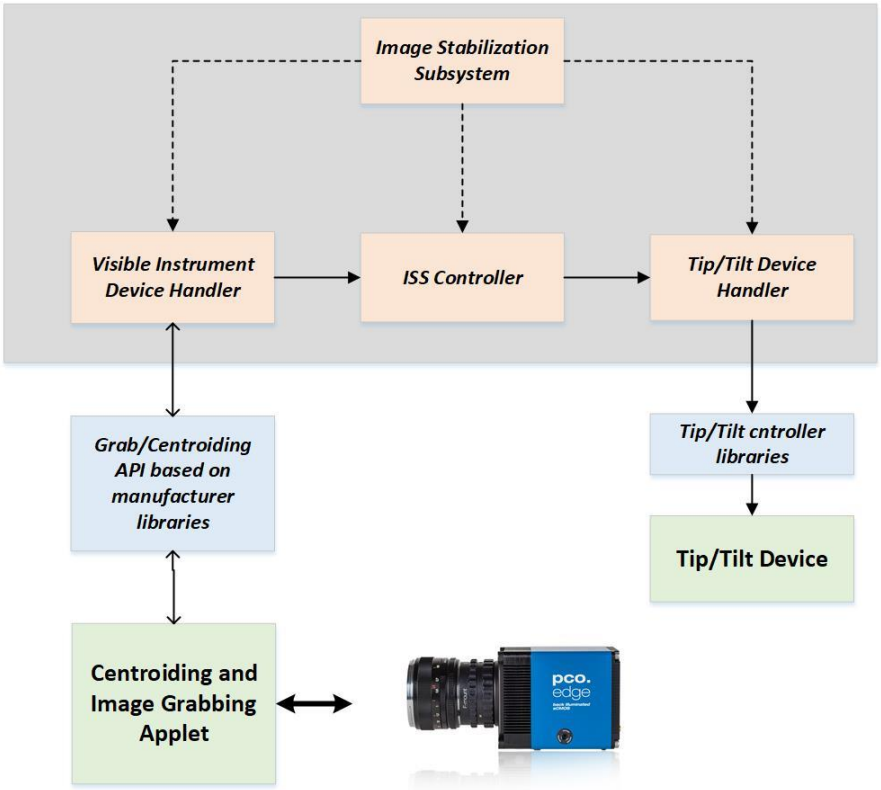

Fig. 12. Breakdown of the telescope stabilization subsystem. Green blocks are hardware and embedded devices, blue is non-framework integrated code and libraries, and pink blocks are mission specific components integrated with FSFW

The development and testing has been performed in three different geographical locations. To reduce the computation load on the flight computer, image-processing algorithms are performed on the Frame Grabber card, which as a FPGA for this purpose. The flow of test and development of this system is shown in Fig. 13. The different teams working on different components are color-coded.

There were many steps that could be avoided, or optimally performed, if a test system for the framework was available from the beginning. However, this has just recently been available, and therefore, we had to test devices and controllers at first with non-FSFW compatible code.

On the other hand, there were some dependencies in base class definitions that have been removed since the beginning of the project. One example was the dependency on power switches in base class. The base class required power switches to be defined for each device upon instantiation to start up toward a functioning state. In our system, we do not have power switches for each device separately and therefore, the FSFW base classes are modified to accept devices without power switches.

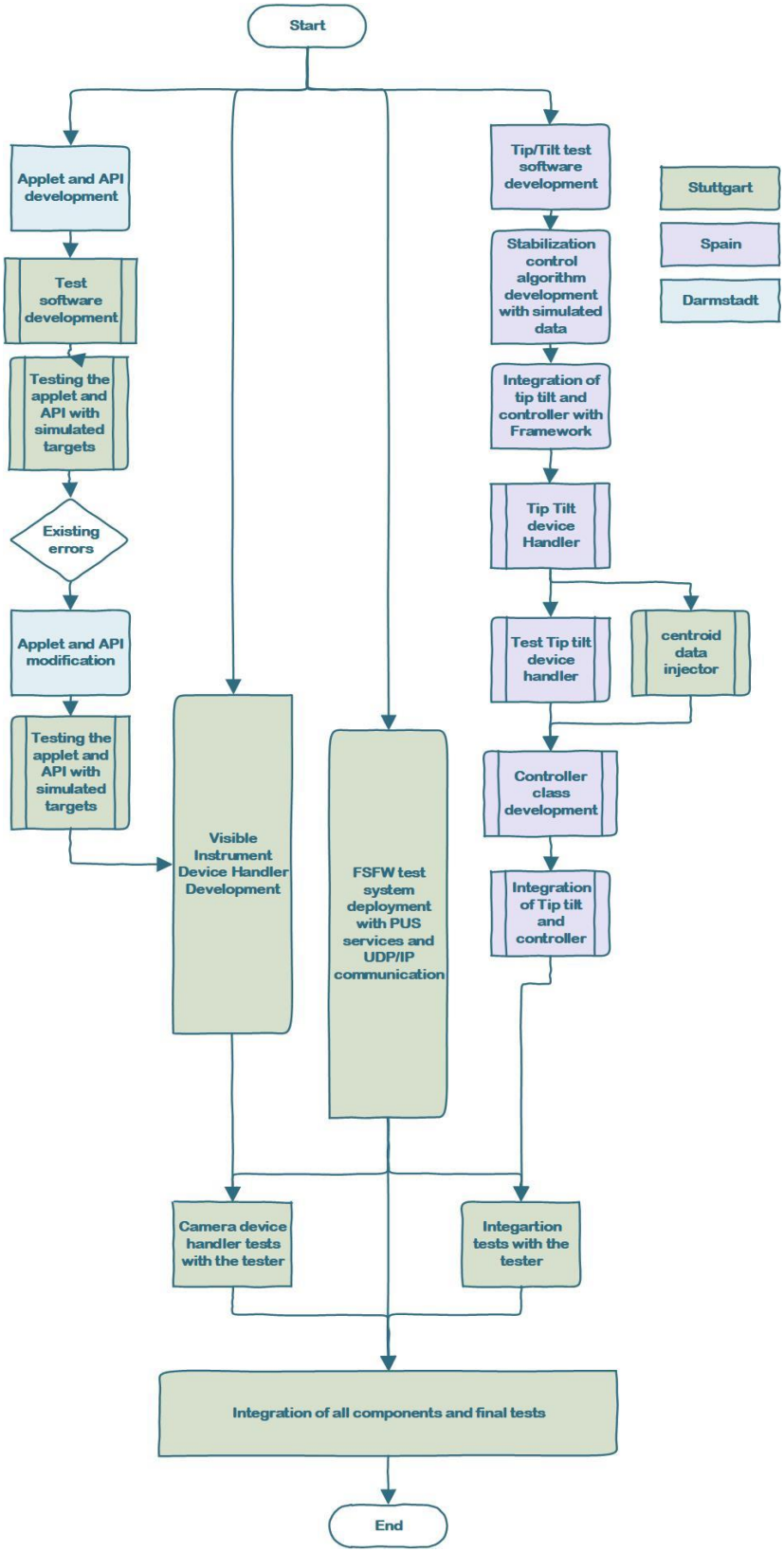

Fig. 13. Development, integration and test process of stabilization system in a distributed environment

6. Conclusions

This paper aimed at looking at the ESBO flight software architecture. The architecture described here is driven by the need for extendibility and reusability of the code for regular flights. ESBO vision is to be able to fly regularly, with different telescopes and different sets of instruments on-board.

Considering the overall design, we have preferred to have a centralized software, and integrate all the instrument-handling codes with our flight software. For this purpose, we have decided to use a core software framework developed for small satellite missions as the core of our flight software. This core software has base classes/templates for devices and controllers which helps the future instrument developers integrate their instrument with the system. It also facilitates modifying the currently implemented components, or modifying the data interfaces with instruments in future.

Our biggest challenges in using a white-box framework has been the learning curve for our developers, and the interdependencies of the components that requires a relatively complex ground test system that could encode and decode space packets, along with part of the PUS service implementation. These components implemented late through the process and it costed us some time in the development process.

With the availability of the test system and some small modifications of the FSFW base classes, we foresee that this architecture can fulfil the flexibility needed from ESBO flight software. Nevertheless, there would still be room to improve the FSFW, to be able to use distributed computing architectures if required in future missions, or to change the base classes to include devices with third party libraries easier.

Acknowledgements

ESBO DS has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 777516.

The authors thank Sabine Klinkner, Ulrich Mohr, Steffen Gaißer, Robin Müller, and all the developers of FSFW for their valuable support and inputs during the development of ESBO flight software.

In addition, the authors also appreciate the support of the students that were involved in the development and the collaboration with the rest of the ESBO DS project team.

References

[1] A. Cho, Cheap balloon-borne telescopes aim to rival space observatories, Science Magazine, 27.02.2020, https://www.sciencemag.org/news/2020/02/cheap-balloon-borne-telescopes-aim-rival-space-observatories.

[2] C. Cofield, NASA Mission Will Study the Cosmos With a Stratospheric Balloon, NASA, 23.07.2020, https://www.jpl.nasa.gov/news/news.php?feature=7712

[3] N. Galitzki, P. Ade, F. E. Angilè, P. Ashton, J. Austermann, T. Billings, … P. Williams, Instrumental performance and results from testing of the BLAST-TNG receiver, submillimeter optics, and MKID detector arrays, Proc. SPIE 9914, 99140J (2016).

[4] P. Maier, J.L.Ortiz, … L. Alvarez Cuevas, ORISON, a stratospheric project, European Planetary Science Congress, Riga, Latvia 17-22 September 2017.

[5] P.Maier, M. Ångermann, J. Barnstedt, S. Bougueroua, A. Colin, … J. Wolf, Stratospheric Balloons as a Complement to the Next Generation of Astronomy Missions,71st Interntational Astronautical Congress, 12-14 October 2020.

[6] J.-E. Strömberg, A Modular Solution for Science Platform Stabilisation, Proc. 20th ESA Symposium on European Rocket and Balloon Programmes and Related Research, Hyère, France ,22 – 26 May 2011.

[7] A. Pahler, M. Ångermann, J. Barnstedt, S. Bougueroua, A. Colin, … J. Wolf, Status of the STUDIO UV balloon Mission and platform, Proc. SPIE 11445, Ground-based and Airborne Telescopes VIII, 114451Y (2020).

[8} B. Baetz, U. Mohr, K. Klemich, N. Bucher, S. Klinkner and J. Eickhoff, The Flight Software of Flying Laptop: Basis for a reusable Spacecraft Componente Framework, IAA Symposium, Berlin , 2017.

[9] B.Baetz, Design and Implementation of a Spacecraft Flight Software Framework, Ph.D. thesis, Institute of Space Systems University of Stuttgart, Pfaffenwaldring 29, 70569 Stuttgart, 2018.

[10] J. Eickhoff, B. Chintalapati1 Stoecker, W. von Kader, R. Traussnig, C. Sayer , R. Peel, The Flexible LEO Platform For Small Satellite Missions, Deutscher Luft- und Raumfahrtkongress 2018