SSC23-WP2-05

Design of the Hosted Software Application for the VISORS Mission

Ebenezer Arunkumar, Antoine Paletta, Glenn Lightsey
Georgia Institute of Technology
620 Cherry St NW, Atlanta, GA 30332
earunkumar3@gatech.edu

ABSTRACT

The VIrtual Super Optics Reconfigurable Swarm (VISORS) mission is a distributed space telescope consisting of two 6U CubeSats that utilize precision formation flying to detect and study the fundamental energy release regions of the solar corona. The inherent complexities and risks associated with two spacecraft operating in close proximity, as well as the unique restrictions of the spacecrafts' design, make careful autonomous execution crucial to the success of the mission. To address these challenges, this paper outlines the development of the Hosted Software Application (HSA) flight software which manages the Guidance, Navigation, and Control (GNC) algorithms, the payload finite state machine, and the spacecraft and formation level fault management system. An overview of the HSA provides context for the motivation and requirements driving the design of the flight software system. The architecture of the HSA is presented and shown to be derived from the Mission Events Timeline (MET) for each of the relevant phases of the mission. Finally, this paper briefly discusses the software's implementation and test campaign.

INTRODUCTION

As the overall number of space launches increases, the number of rideshare opportunities has also increased, directly contributing to the increase in CubeSat launches from under 100 cumulatively by 2013 to over 1800 by 2022.[1] Increased access to space has enabled many entities, such as higher education universities, to enter the small satellite development space. However, developing small satellites comes with many challenges, especially at the university level. Even though launch opportunities have decreased in cost, funding issues, along with a lack of overall experience and a high turnover rate of students, lead to projects that are rushed, lack redundancy, and lack adequate testing. These issues permeate to the subsystem level and affect their reliability and robustness.

The lack of student experience and knowledge is especially apparent in flight software (FSW) development, as FSW is often considered a 'black box' due to how difficult it can be to develop. At the university level, the tools and framework to start writing FSW are often non-existent. This results in missions that have to develop tools and infrastructure before they can even begin development. This additional work further exacerbates the issues many small satellite missions face and leads to additional cost and schedule overruns. One way to mitigate this issue is to use software frameworks that provide much of the basic development tooling and design the software to be as simple, yet robust, as possible. Reducing the complexity of the design eases the burden on software developers but also simplifies the lives of mission operators who, more often than not, were not involved with the development of the FSW. As a result, spending a significant amount of time meticulously planning out a software design can decrease overall development time and help university-level teams stick to their tight schedules.

However, designing a FSW system from scratch is extremely difficult. For a formation flying mission, such as the one presented in this paper, the flight software architecture becomes even more complex as additional subsystems pertinent to formation flying are introduced, including formation level fault detection, inter-satellite link, relative propulsion, and distributed science instruments. The FSW must also handle any concurrency issues that arise when controlling two independent spacecraft that must work together, such as ensuring that both spacecraft do not have different formation state information. Instead of immediately diving into the specifics of the flight software itself, this paper highlights a design approach which first looks at the concept of operations (CONOPS) and mission events timeline (MET) for every phase of the mission. By first determining what each spacecraft needs to do to accomplish the goals of the formation, the specific tasks that the FSW must complete can be more granularly identified. This inherently leads to FSW requirements that drive how the software should be architected to complete those tasks.

The goal of this paper is to illustrate the design of the Hosted Software Application (HSA) flight software on the VISORS mission. By discussing the development lifecycle of the HSA, this paper intends to serve as a

roadmap for students who may develop similar software systems for other formation flying or single spacecraft missions. First, an overview of the VISORS mission is given, with an emphasis on the aspects of the mission that the HSA manages. Next, an overview of the goals and requirements of the HSA is discussed. The paper then details how the design of the HSA was derived from the MET for each phase of the mission. Finally, the implementation and testing of the HSA are briefly discussed.

VISORS MISSION OVERVIEW

The goal of the VISORS mission is to detect and study the fundamental energy-release regions of the solar corona. The mission achieves its science goals with a distributed space telescope consisting of two 6U CubeSats, the Detector Spacecraft (DSC) and the Optics Spacecraft (OSC) as seen in Fig 1. The spacecraft buses are nearly identical commercial off the shelf (COTS) XB1 buses provided by Blue Canyon Technologies (BCT). The XB1 provides the spacecraft chassis, the main flight computer, the attitude determination and control (ADCS) system, and the ultra-high frequency (UHF) space-to-ground communication architecture. The rest of the spacecraft is comprised of the payload, including a cold-gas propulsion system (PROP), inter-satellite communications hardware (XLINK), a payload avionics interface board (PAIB), and science instrumentation. Each of the payload subsystems was designed and developed by one of the 11 different institutions on the VISORS mission. VISORS is funded by the National Science Foundation (NSF) and was originally devised at the CubeSat Ideas Lab in 2019.

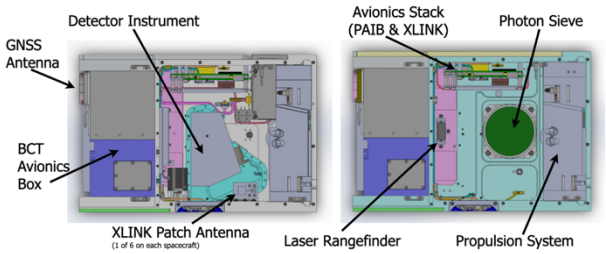

Figure 1. CAD of Detector Spacecraft (left) and Optics Spacecraft (right)[2]

The VISORS mission is officially classified as a technology demonstration mission, focusing on the design and development of novel technologies such as differential carrier-phase global navigation satellite system (GNSS) navigation, inter-satellite communications, and a cold gas propulsion system for high precision relative maneuvering.[3,4] These technologies work together to enable the VISORS spacecraft to maneuver to a relative separation of 40 meters during a science observation, the focal length required to obtain images of the sun in the He II 304 Å line, as shown in Fig 2. These novel technologies also enable the mission to meet its minimum science success goal of obtaining a single image of the sun in this He II 304 Å wavelength with a resolution of 0.2 arcseconds.[2]

The science instrumentation that enables these images are distributed across the two spacecraft.[2] The OSC contains the extreme ultraviolet photon sieve optic which focuses incoming light onto the detector instrument located on the DSC. The OSC also contains a laser rangefinder (LRF) that is used for different purposes by the guidance, navigation, and control (GNC) algorithms and the science team on the ground. During an observation attempt, the GNC algorithms have the ability to autonomously use the LRF data to help control the formation into the observation alignment. After downlinking the LRF telemetry, the science team uses the timestamps on the ranging data to help inform their decision on which images they want to downlink. On the DSC, the detector instrument contains the Compact Spectral Image Electronics (CSIE) which controls the camera and processes the images before preparing them

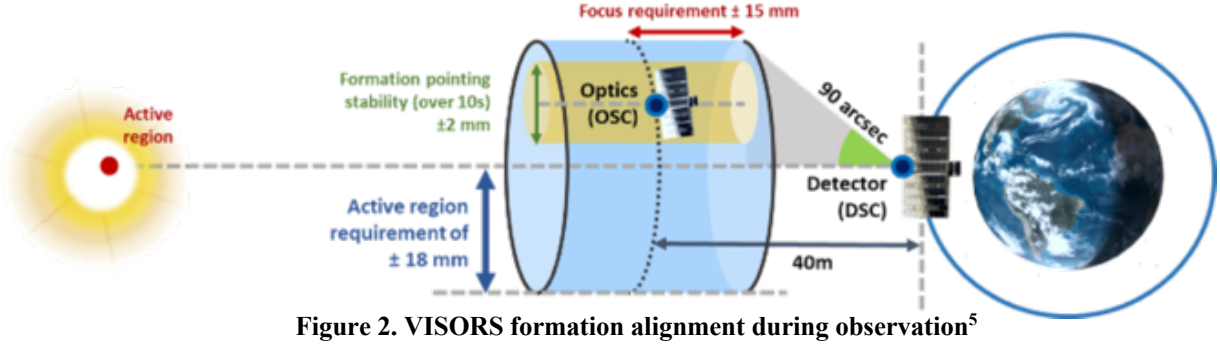

Figure 2. VISORS formation alignment during observation[5]

for downlink. While the science instrumentation differs between the two spacecraft, many of the payload subsystems are nearly identical. This design decision simplifies overall system development, as well as the spacecraft integration and test campaign.[6] The VISORS mission is currently awaiting the delivery of flight payload subsystems and the DSC bus, shown in Fig 3. The integration and test campaign for both spacecraft will begin in the fall of 2023, with a projected spacecraft delivery date of July 2024.

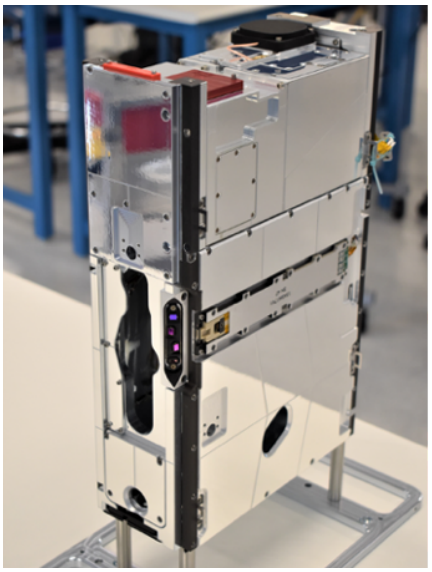

Figure 3. Detector Spacecraft during testing at BCT[7]

GNC Overview

At a high level, the GNC system on the VISORS mission has two goals. First and foremost, since VISORS is a formation flying mission, the GNC algorithms must ensure that the spacecraft avoid collisions. This is done passively through the design of the relative orbits and actively through preemptive collision avoidance maneuvers. The passive safety margin guarantees that the spacecraft will not collide for at least 2 orbits in the absence of maneuvers. The collision avoidance maneuvers aim to increase the relative separation in the radial-normal plane of the radial-tangential-normal (RTN) frame while introducing relative drift in the tangential frame. The second main goal of the GNC algorithms is to control the formation to millimeter-level position accuracy and micrometer per second-level velocity accuracy to ensure that the science instruments can take images that are in focus, on target, and have acceptable smearing. These GNC goals are captured in the VISORS mission objectives shown in Table 1 and discussed in more detail in Ref 3.

Table 1. VISORS Mission Objectives (GNC-related objectives in bold)

Identifier	Objective
MO-001	Capture and downlink coronal imagery to determine the existence of energy-release regions in the solar corona
MO-002	**Control formation to millimeter-level position accuracy**
MO-003	Inter-satellite communication link enabling autonomous maneuver planning
MO-004	**E/I-vector separation to enable passive collision avoidance and maintain near-proximity relative orbits**
MO-005	Propulsion systems for formation-keeping and reconfiguration

Staying in a 40-meter relative separation formation configuration for the whole mission is risky and expensive from an energy and delta-v standpoint. As a result, the GNC algorithms have defined multiple orbit configurations that will occur over the course of the mission. These orbits are defined in Table 2. In addition to these pre-defined relative orbits, the spacecraft may be in a configuration with *no* relative orbit (for example during commissioning or after an escape maneuver). The design of the GNC algorithms is such that for all of these relative orbits, only one spacecraft is maneuvering at a time. The active spacecraft (the deputy) performs relative maneuvers about the passive spacecraft (the chief).

Table 2. Mission Defined Relative Orbits

Relative Orbit	Description
Science Orbit	Ellipse with nominal relative separation of 40 meters
Standby Orbit	Minimum relative separation of 200 meters
Transfer Orbit	Relative orbit trajectory to reconfigure the formation between the science and standby orbit

CONOPS Overview

The design of the mission concept of operations (CONOPS) was driven by the various mission-defined relative orbits described in the previous section as well as the constraints imposed by being a payload of the COTS XB1 spacecraft. The CONOPS can be separated into two main sections – the spacecraft state diagrams and the mission events timeline. The spacecraft finite state machines detail the logical conditions of the spacecraft and formation and define the entry and exit criteria of each state.[8] On the other hand, the MET outlines the specific spacecraft and formation-level actions that occur during each phase of the mission. The VISORS MET's are discussed in more detail in a latter

section. For clarity, nomenclature related to the VISORS state machines is given in Table 3.

Table 3. VISORS CONOPS Nomenclature

Nomenclature	Description
Spacecraft Modes	BCT defined states for the COTS XB1 spacecraft
Mission Modes	VISORS specific states of the formation
Subsystem States	Individual 'state' of each payload subsystem (ON or OFF)
Spacecraft Role	Delineation of which spacecraft is the active, maneuvering, spacecraft and which one is the passive, non-maneuvering, spacecraft

The first set of states for the VISORS spacecraft, the spacecraft modes, are shown in Fig 4. These modes are defined by BCT and are standardized for the XB1 spacecraft. When the spacecraft is initially launched, it will boot up in Launch Mode and then autonomously transition into Sun Point Mode at the conclusion of a 30-minute deployment timer. In Sun Point Mode, the spacecraft slews to point its solar panels toward the sun while only keeping BCT subsystems powered on. No payload subsystems are turned on until a ground command is given to put the spacecraft into Fine Reference Point (FRP) mode. In the event that the battery voltage drops below a critical threshold, the spacecraft will go into Survival Mode from either Sun Point or FRP modes for the sole purpose of charging its batteries. BCT restricts the payload to be off in Sun Point and Survival modes to prevent unnecessary power draws. This restriction, along with the fact that the BCT state machine cannot be modified, led to the creation of a second, mission-specific, state machine.

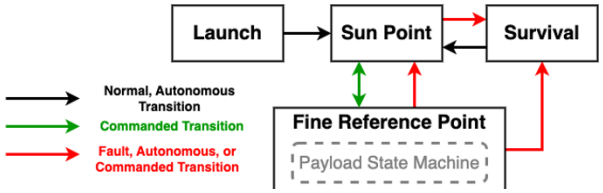

Figure 4. BCT Defined Spacecraft Modes[9]

When the spacecraft is in FRP mode, the payload operates within the constraints of the mode diagram shown in Fig 5. During the early phases of the mission while the ground operators are performing payload commissioning, the payload will stay in its preliminary operations mode. Once all preliminary operations have been completed, the spacecraft will transition into the first of three nominal mission modes, Standby. Since the GNC algorithms have defined three main relative orbits for nominal operations, it follows that each of these orbits correlates one-to-one with a payload mission mode. During a nominal science campaign, the payload will transition from Standby mode to Science mode via Transfer mode. The payload also contains two off-nominal mission modes – Escape and Safe modes. If there is a collision risk, hardware payload fault, or software payload fault, the payload will transition into either the Escape or Safe mode, depending on the nature of the fault. If the payload first enters into the Escape off-nominal mode, it will autonomously transition into Safe mode after an escape maneuver is performed. On the other hand, if the payload first enters Safe mode, it will stay in Safe mode unless it has to perform an escape maneuver (which would send it into Escape mode and then back into Safe). Note that the payload is not allowed to perform more than one escape maneuver in sequence in order to prevent the spacecraft from triggering maneuvers that could potentially further harm the safety of the formation. Ground operator intervention can restore the spacecraft's ability to go into Escape mode as well as allow the payload to return to its nominal mission modes from Safe mode. This ensures that the spacecraft only return to nominal operations after a thorough analysis of what caused the spacecraft to enter Safe mode in the first place.

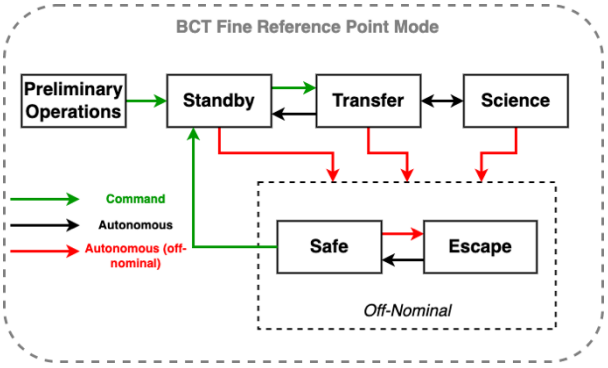

Figure 5. Payload Defined Mission Modes

In each of the payload mission modes, there is a parameterized configuration of payload subsystem states. This configuration defines which payloads are operational during each mission mode and ensures that no subsystem state is set to ON unless it is required in that mode. However, since VISORS is a formation flying mission, all payload subsystems except the science instrumentation are critical to maintaining the formation's relative configuration. As a result, most of the payloads will always be ON during every payload mission mode unless there is an off-nominal condition. In addition, if the spacecraft ever exits the FRP spacecraft mode all payload subsystems are powered off by BCT.

HOSTED SOFTWARE APP REQUIREMENTS

Each of the subsystems onboard the two spacecraft have hardware specific software systems to manage

subsystem level tasks. This simplifies the overall software system of the spacecraft as there are now discrete software packages for each subsystem instead of one large software package for all subsystems. For example, the propulsion software is solely responsible for the command and telemetry interface for the propulsion hardware and firing thruster valves. Similarly, the CSIE subsystem is solely responsible for taking science observations. However, during the initial design of the VISORS mission, it was apparent that additional higher-level on-board software was needed for four reasons:

1) To host the GNC algorithms
2) To manage the payload finite state machine
3) To monitor and respond to payload faults
4) To communicate with all payload subsystems

To meet the four goals shown above, the software system needed to interface with all payload subsystems. However, the only hardware subsystem that had data interfaces with all subsystems was the XB1 flight computer. Thus, it was decided that the additional software – the Hosted Software Application - would be hosted on an independent partition of the flight computer. The following sections detail the background for each of the four goals of the HSA and outline the level two requirements that are derived from those goals. These requirements help identify the various modules of software that are required for the HSA to perform its tasks.[10] Aside from the requirements that come from these four goals, additional requirements are also outlined that stem from the HSA being a software payload on the BCT XB1 flight computer.

HSA-GNC Interaction

Due to the complex nature of the GNC software, it was determined from an early stage that the GNC algorithms should be a separate module of software from any other software system on the spacecraft. However, due to the computational and memory constraints of the microcontrollers on payload subsystems, the GNC algorithms needed to execute on a more capable processor. To avoid changing the design of any of the payload subsystems to include a more powerful processor, the BCT XB1 flight computer was chosen to run the GNC algorithms since it met the computational requirements. Since the XB1 can only support one payload software subsystem, the Stanford team delivered the GNC algorithm as a C++ library that can be compiled into the HSA executable. To enable this architecture, the HSA includes a module of software called the **GNC Controller** that provides any inputs that the GNC library needs and accurately responds to the outputs from the GNC library. The GNC Controller also enables the GNC algorithms to run on its own thread so that the tasks of the rest of the HSA never interfere with the run-time of the GNC algorithms. The requirements outlining the GNC interaction with the HSA are documented in Table 4.

Table 4. HSA Requirements related to the GNC Subsystem

ID	Requirement
HSA-002	The HSA shall include the GNC library in the compiled executable.
HSA-003	The HSA shall interface with the GNC library in accordance with the GNC ICD
HSA-010	The GNC Library within the HSA shall run on its own thread.
HSA-011	All checksum validated data received by the HSA from the ISL shall be immediately forwarded to the GNC library.
HSA-021	The HSA shall parse the information contained in the time at tone packet received from the BCT Bus and deliver it to the GNC Software library.

VISORS Finite State Machine

To manage the payload mission modes, spacecraft formation role, and subsystem states, the HSA includes a software module called the **Payload State Machine** (PSM). Since the mission mode and subsystem states are inherently tied together, only one software module was needed to control both items instead of separating control into distinct modules. For conciseness and simplicity, the spacecraft role – active or passive – is also managed by the PSM even though it is not directly tied to a mission mode or subsystem state (since either spacecraft can be active or passive at any point).

The PSM can manipulate the modes, states, and roles of the spacecraft and formation in three different ways – via ground command, via predetermined nominal operations configurations, and via off-nominal fault response operations. Firstly, the PSM can always be commanded to change any modes, states, or roles via ground command. Regardless of what is occurring internally on the spacecraft, commands from the ground always take precedence over any autonomous actions. This decision was made so that ground operators could retain full manual control in case on-board autonomy did not behave as expected. Secondly, during nominal operations the PSM changes its modes based on interactions with the GNC Controller; once the GNC algorithms have transitioned between the various mission-defined relative orbits, the PSM will be alerted to change into the corresponding mode. During each of the nominal modes, the subsystem states are toggled based on the parameterized configuration that corresponds to each mission mode. During nominal operations, the spacecraft's role does not change and is set to whatever the ground operators designated at the

beginning of the mission. Finally, the PSM can change any mode, state, or role via recommendations from the fault management system. Depending on the type of fault – collision risk, hardware fault, software fault – the PSM will change the mission mode, subsystem states, or spacecraft role.

State machines on formation flying missions have another layer of complexity as the mission must decide how they want to address congruency in the states between the spacecraft. For the VISORS mission, congruency is achieved by ensuring that both spacecraft must always attempt to be in the same payload mission mode as the other spacecraft. This means that the spacecraft must always be interchanging their mission modes over the XLINK system. However, while symmetry in states between spacecraft is preferred, there are a few operational scenarios where the spacecraft are not in the same mission mode. For example, if the active (maneuvering) spacecraft goes into Escape mode the passive (non-maneuvering) spacecraft will automatically transition into Safe mode since it is not performing a maneuver. This ensures that both spacecraft do not perform escape maneuvers at the same time. The HSA requirements surrounding the PSM are found in Table 5.

Table 5. HSA Requirements related to the Payload State Machine

ID	Requirement
HSA-004	The HSA shall include a state machine to actuate mission modes and subsystem states in accordance with the Subsystem States Document.
HSA-018	The HSA shall send commands to the PAIB to toggle power to payload subsystems.

VISORS Fault Management Overview

On most spacecraft, fault management systems are autonomous systems that strive to detect, isolate, and recover from any situations that upset nominal operations.[11] These systems are often the products of failure modes and effects analysis that attempt to understand the different failure modes and what the resulting fallout would be for each scenario.[12] For formation flying missions such as VISORS, additional failure modes must be considered such as the risk of collision between spacecraft. This additional failure mode results in a contingency operations architecture that must manage subsystem and formation-level faults.[13] Since many of the subsystems on a formation flying mission are dedicated solely to enabling safe formation flight, subsystem level and formation level faults are often coupled, potentially complicating the autonomous fault response.

To correctly diagnose and respond to faults on VISORS, the fault management system is split into two distinct software modules - **Fault Detection** (FD) and **Fault Response** (FR).[14] The FD module on each spacecraft is responsible for monitoring the telemetry points from all subsystems on both satellites and determining if any of them have crossed their nominal thresholds. By monitoring which telemetry fields go out of bounds, the FD module can diagnose the problem and understand what caused it. The key to the diagnosis is the fact that the FD modules have full state knowledge on the health of each spacecraft and thereby the health of the formation. The main challenge for the fault detection module is that monitoring one telemetry point alone is often not enough information to deduce the health of a subsystem or the formation. Telemetry points from multiple different sources must be analyzed to understand whether the spacecraft is experiencing a fault and to determine the source of that fault.

Once the fault has been diagnosed, the FR module must decide the appropriate action to take to mitigate the fault. On the VISORS mission, fault responses consist of combinations of a mission mode switch, a subsystem power cycle or shut down, or a spacecraft role switch. Role switches are included as a fault response to take advantage of the fact that both spacecraft have maneuvering capability. Thus, in the event of a failure of the propulsion system on one spacecraft, the formation has redundancy and can continue its operation by allowing the other spacecraft to assume the active role. Regardless of the fault diagnosis passed into the fault response module, the response is always chosen to be as conservative as possible. This keeps the logic simple and ensures that the spacecraft can make any required autonomous decisions to keep the formation safe, while also waiting for ground operators to handle more complex formation-level responses.[14]

To support the fault management system, a few additional software modules are required. For example, since the telemetry packets from other subsystems come in as serialized data, a parser component – the **Telemetry (TLM) Parser** - is necessary to unpack each of the subsystem data packets. Since the parsing of packets is only necessary for the fault management system, the TLM Parser only deserializes the specific telemetry fields necessary for fault detection. After all desired data is received and unpacked, it must be stored in a database for later retrieval by the FD module. This database software module is named the **Polymorphic (Poly) Database** as it stores telemetry fields in their native type instead of in serialized form. The HSA requirements related to the fault management system are found in Table 6.

Table 6. HSA Requirements related to the Fault Management System

ID	Requirement
HSA-005	The HSA shall include a Fault Detection and Response Block to monitor all fault scenarios specified in the Fault Analysis Matrix.

VISORS HSA Communication Overview

Since the HSA is hosted on the XB1 flight computer, its interface must comply with the specifications of the XB1 spacecraft. The XB1 requires that all payload subsystems, including the HSA, communicate using the Consultative Committee for Space Data Systems (CCSDS) Space Packet Protocol. This protocol specifies that every software data packet contains at least a 6-byte header that contains packet version numbers, packet identification, packet sequence number, and packet data length.[15] As a result, the HSA must have the functionality to frame any outgoing data packets and deframe any incoming data packets per the CCSDS protocol. This functionality is encapsulated in two different software modules – the **CCSDS Framer** and **CCSDS Deframer**.

To route the data to the correct location during the framing and deframing step, each packet contains a unique packet identification number, often called the APID. To communicate digitally with the BCT Bus FSW, as shown in Fig 6, the HSA contains a software module - IPC Driver - that uses the inter-process communication protocol.[16] The HSA requirements that relate to the software interfaces with the rest of the payload are listed in Table 7.

Table 7. HSA Requirements Related to SW Interfaces

ID	Requirement
HSA-008	The HSA shall send all generated telemetry to the BCT Bus radio downlink buffer using the interface specified in the BCT SW API.
HSA-015	The HSA shall interface with the BCT FSW using the ports specified in the BCT SW API.
HSA-016	The HSA shall adhere to the CCSDS Protocol Specifications specified in the BCT XB1 ICD when communicating with the spacecraft bus.
HSA-017	The HSA shall adhere to the APID ranges specified in the BCT XB1 ICD (BUS-EC-001) for communication with other subsystems.

Additional HSA Requirements

In addition to the modules discussed above, several additional modules are necessary to provide basic embedded systems functionality. For the HSA to interface with ground operators, software modules to receive commands (**CMD Dispatcher**), send distinct fixed-size telemetry packets (**Downlink Packetizer**), configure parameters for the HSA software (**Parameter Database**), and log software event verification records (EVRs) (**Event Logger**) are included within the software executable. Additionally, the HSA includes software modules that enable a robust implementation and execution of logic such as modules for continuous execution loops of rate groups (**Linux Timer**, **Rate Group Driver**, **Active Rate Group**), data buffer managers (**Buffer Manager** & **Static Memory**), assert handling (**Fatal Adapter** & **Fatal Handler**) and time correlation based off of the XB1 system clock (**Linux Time**). Finally, the HSA contains a module - **System Resources** - to characterize the resource utilization of the HSA to ensure it stays under requirements HSA-012 and HSA-013 found in Table 8.

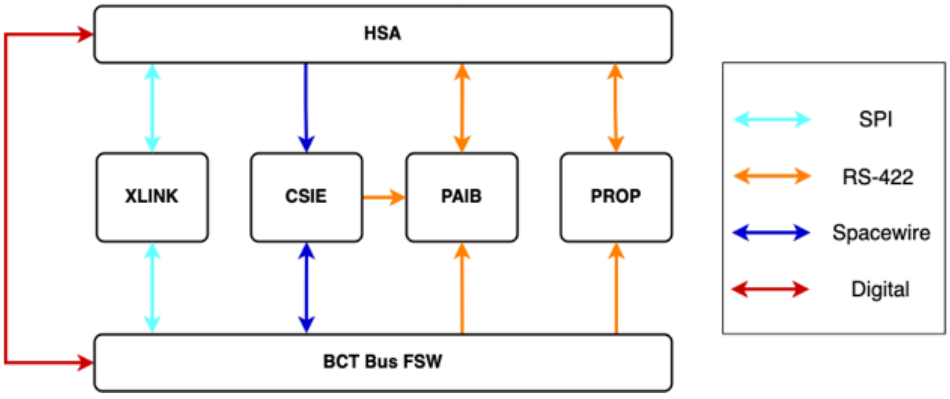

Figure 6. Payload Subsystem Interface Diagram

Table 8. Miscellaneous HSA Requirements

ID	Requirement
HSA-001	The hosted software application (HSA) shall compile into a single executable in accordance with the BCT SW API.
HSA-007	The HSA shall generate a telemetry packet at a rate specified in the data budget.
HSA-012	The HSA shall not take up more than 4 MB of program memory per BCT XB1 ICD (BUS-EC-001).
HSA-013	The HSA shall not take up more than 60 MB of RAM per the BCT XB1 ICD
HSA-014	The HSA shall have a method to determine which spacecraft it is running on.

The complete list of the software modules found in the HSA is shown in Table 9. The breakup of scope between modules could have been done in multiple different ways, depending on the system architect. However, the guiding principle for this specific delineation of software modules was the separation of concerns philosophy. Separation of concerns states that the software system should be decomposed into modules that each discretely solve the different aspects of the problem.[17] This principle lends itself well to the component-based architecture of the Fprime software framework, discussed in the next section.[18] Additionally, as shown in Table 9, many of the modules were provided and built into the Fprime framework, reducing overall development time.

Table 9. List of all modules in the HSA

Component Name	Description	Development Type
CMD Dispatcher	Distributes commands to all components	Built-In
Parameter Database	Store non-volatile parameters used by any component	Built-In
TLM Database	Stores telemetry generated by any component	Built-In
Event Logger	Log flight software 'events' for greater insight into the FSW execution	Built-In
Linux Timer	Output a constant tick to the RG components at a specified time interval based off of the system clock	Built-In
Rate Group Divider	Divide constant tick from linux timer into ticks for each rate group	Built-In
Rate Group Component	Distribute rate group calls to other components at correct rate	Built-In
Linux Time	Correlate software timestamps to system time	Built-In
Buffer Manager	Manage memory allocation for components using dynamic buffer sizes	Built-In
Static Memory	Manage memory allocation for components using fixed buffer sizes	Built-In
Fatal Adapter	Intercept assert calls and log corresponding fatal events	Built-In
Fatal Handler	Handle fatal events by delaying segmentation fault by one second to allow for fatal events to propagate to the ground system	Built-In
System Resources	Track resource utilization of CPU and RAM	Built-In
Poly Database	Store telemetry values from subsystems that pertain to fault detection	Built-In
CCSDS Framer	Pack outgoing data packets into the CCSDS Format	Custom
CCSDS Deframer	Unpack incoming CCSDS data packets to retrieve desired data	Custom
IPC Driver	Communicate with the BCT Bus FSW using IPC protocol	Custom
TLM Parser	Unpacks desired telemetry fields from subsystem telemetry packets	Custom
Downlink Packetizer	Forms fixed size data packets containing HSA telemetry	Custom
Fault Detection	Detect and diagnose payload level fault conditions	Custom
Fault Response	Choose appropriate payload response based off fault diagnosis	Custom
Payload State Machine	Control mission mode, subsystem state, and formation role	Custom
GNC Controller	Provide wrapper for the GNC algorithm library	Custom

HOSTED SOFTWARE APP FRAMEWORK

With all required software modules defined, the design of the HSA can be developed. The first step to starting development is to choose the flight software architecture. The main driver for choosing the software architecture for the HSA was picking a framework that would best enable fast and robust development, while also providing tools for developers and operators for testing and operations. The most common choices for software frameworks for CubeSat-level missions are either cFS (core flight software) developed at NASA Goddard, Fprime developed by NASA JPL, or a custom built-from-scratch framework.[19, 20] For the VISORS mission, all GT software was written within the Fprime framework (v3.1.1) due to prior experience among the members in the lab as well as existing development tools that were developed for previous missions.

Fprime is an open-source C++ framework that was initially released to the public in 2018. It is a component-based point-to-point architecture that enables modularity and reuse of software.[20] Fprime ships with several ready-to-use components that are found on most embedded

systems projects such as a Command Dispatcher, Parameter Database, and Telemetry Database, among others. Table 9 shows that about 60% of the components were already available within the Fprime framework while about 40% were custom designed for the VISORS mission. Fprime also provides supporting tools to speed up development, testing, and operations, such as a custom ground data segment (GDS) as well as several all-inclusive autocoders that autogenerate large swaths of code based on simply the input and output interfaces to each component.[21]

The Fprime architecture consists of three main parts – Ports, Components, and Topologies. Each software module, or component, consists of input and output ports that define the data structure. The component has handler functions for each of the input and output ports that define the logic that must be executed to send or receive data. Additional functions can also be defined in the component C++ code. Components are then hooked up to other components within the topology. The topology, such as the one shown in in Fig 7, ultimately provides an overarching view of the design of the software executable, called a deployment.

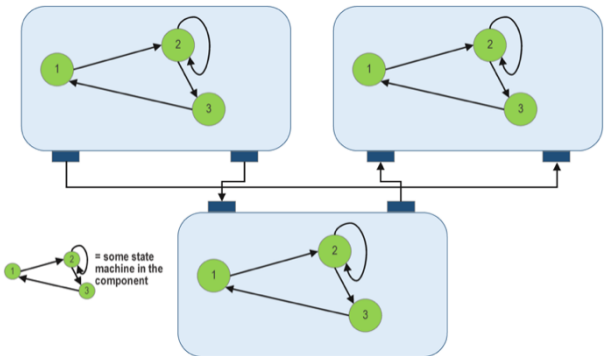

Figure 7. F' Component-Based Architectural Pattern[18]

The one obvious drawback to having so many discrete components in a deployment is that connecting them together in the topology can be confusing for large deployments that require numerous port connections. However, another huge benefit to using Fprime is that for many of the built-in components, Fprime either connects them in the topology automatically or provides clear documentation on how they should be connected to other components. Thus, the more complicated aspect of the topology becomes defining the specific data types that need to be exchanged between components. By looking at the MET for each phase of the mission, the software architect can outline exactly what data must be transferred between components. This definition of the input and output interfaces of each component then corresponds directly to the set of Fprime ports on each component.

HOSTED SOFTWARE APP OPERATIONAL USAGE

To provide context for how the software modules interact with each other, the HSA-related CONOPS and mission events timelines for every phase of the mission must be defined. The entire VISORS mission can be split into distinct phases, as seen in Table 10, each of which can be further characterized with a MET to describe the specific actions the HSA takes during each of the phases. Many of the FSW actions require the use of multiple components since the scope of each component is limited. Thus, outlining each of the FSW actions will directly derive how the components should be connected.

Table 10. HSA status in each phase of the mission

Phase	Description	HSA Status
Launch + Solar Array Deployment	The Spacecraft wait for 30 minutes before deploying solar arrays and turning on all of the BCT subsystems	OFF
Initial Bus Commissioning	Ground-based commissioning campaign to verify the functionality of the BCT subsystems	OFF
Payload Commissioning	Ground-based commissioning campaign to verify the functionality of the payload subsystems	ON
Formation Acquisition	Manually command the spacecraft into the standby formation configuration	ON
Standby	Wait in standby formation configuration for the command to start a science campaign	ON
Science Campaign	Go from the standby to science orbits and take science observations	ON
Off Nominal	Any time a spacecraft detects a fault and must respond	ON

The following subsections detail the HSA-oriented MET for all phases of the mission for which the HSA will be ON. Each subsection will then outline the port connections necessary to achieve each step of the MET. These port connections between the components can be

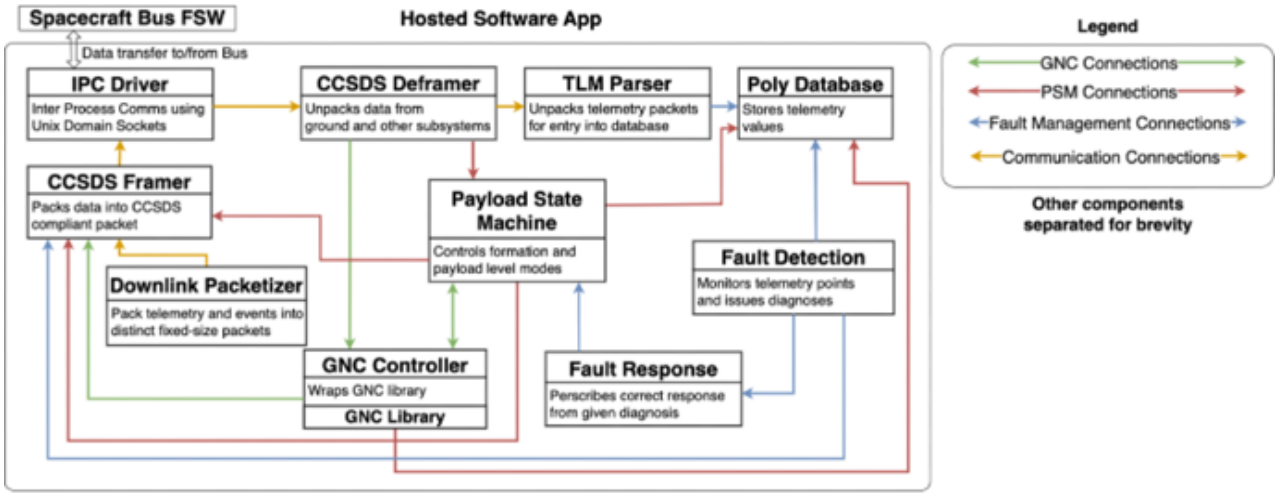

Figure 8. HSA Topology without most of the Fprime built-in components

seen in the simplified topology diagram shown in Fig 8. This diagram includes most of the custom components within the deployment but leaves out many of the built-in components for brevity.

Payload Commissioning

After the BCT Bus hardware subsystems have been commissioned, the next step is to commission the payload subsystems. The HSA portion of the payload commissioning is distributed among the rest of the payload commissioning steps. Since the HSA communicates with all payload subsystems, the HSA interfaces must be checked after turning on any subsystem. This will consist of verifying that commands can be sent to the subsystem and that telemetry is received back from the subsystem. The sequence for this preliminary operations campaign is illustrated in Fig 9. The HSA to BCT Bus FSW interactions must be validated first since all communication with the ground go through BCT FSW.

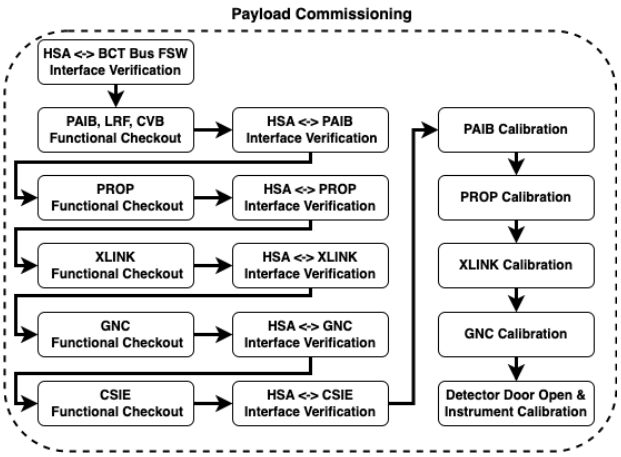

Figure 9. Payload Commissioning Sequence

The HSA to BCT FSW verification step validates the mission operator's ability to send commands to and receive telemetry from the HSA. This step consists of sending the HSA a no operation (NO-OP) command and verifying that it sends out an EVR upon the completion of that command. The NO-OP command is used since it is a simple command that does nothing but acknowledge receipt of the command. The HSA interface verification steps for each of the subsystems are similar to the previous step but delegate the HSA to send the NO-OP command instead of the ground. After the HSA sends out the command it will, depending on the subsystem, either wait for a direct response from the subsystem or parse its telemetry to see if the subsystem processed the command. The HSA will also verify that it receives telemetry from every payload subsystem.

During the calibration section of commissioning, most of the ground-based commands are routed directly to the desired payload subsystem and do not go through the HSA, the only calibration related to the HSA deployment is during the GNC and Prop calibration steps. During GNC calibration, the ground sends several commands to the GNC Controller which simply passes those commands to the GNC algorithms. Receipt of these commands is captured via EVRs and telemetry packets. The port connections required between HSA components for this phase of the mission are outlined below in Table 11.

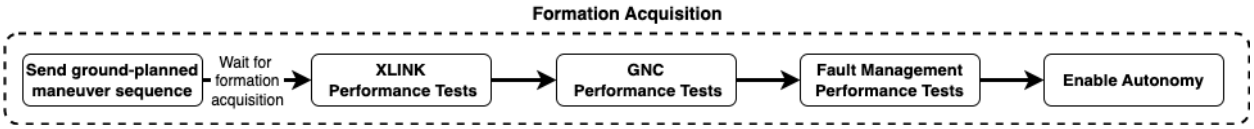

Figure 10. Formation Acquisition Sequence

Table 11. HSA Port Connections Required for the Commissioning Phase

Payload Commissioning Step	HSA Action	Related HSA Port Connection
HSA ←→ BCT Interface Verification	HSA receives NO-OP command from the ground	IPC Driver → CCSDS Deframer → CMD Dispatcher
HSA ←→ BCT Interface Verification	HSA responds to NO-OP command	Downlink Packetizer → CCSDS Framer → IPC Driver
HSA ←→ Subsystem Interface Verification	HSA sends out NO-OP command	Payload State Machine → CCSDS Framer → IPC Driver
HSA ←→ Subsystem Interface Verification	HSA receives confirmation of NO-OP command	IPC Driver → CCSDS Deframer → Payload State Machine OR IPC Driver → CCSDS Deframer → TLM Parser → Poly Database → Payload State Machine
HSA ←→ Subsystem Interface Verification	HSA verifies reception of telemetry from subsystem	IPC Driver → CCSDS Deframer → TLM Parser
GNC Calibration	HSA receives ground commands for GNC Controller	IPC Driver → CCSDS Deframer → CMD Dispatcher → GNC Controller
GNC Calibration	GNC Controller responds to commands with events	Downlink Packetizer → CCSDS Framer → IPC Driver
GNC Calibration	GNC algorithm initiates a propulsive maneuver	GNC Controller → CCSDS Framer

Formation Acquisition

Once commissioning is finished, operators will transition into the formation acquisition phase of the mission. In this phase, the main goal is not only to put the spacecraft into the Standby mode formation configuration but also to verify the performance of subsystems that depend on the spacecraft being in formation. The order of steps in this phase, as seen in Fig 10, is important as the XLINK subsystem needs to be performance tested first since this will enable the testing that follows on the GNC and fault management systems. The final step after the formation is acquired must be to enable autonomy so that the spacecraft can take corrective action in the event of a fault, since the relative dynamics occur faster than a ground response is possible. The duration of this phase must be kept short since the spacecraft will be in standby formation without an autonomous way to escape until autonomy is enabled by the mission operators at the end of the phase.

The main new port connections found in this mission phase are related to the fault management system. The design of the fault management system lends itself to a linear data flow between its components. The fault detection component passes its diagnosis to the fault response component which passes recommended actions to the payload state machine. Autonomous fault responses are actuated through the payload state machine. This segmented approach, seen in the port connections outlined in Table 12, fulfills the separation of concerns guideline.

Table 12. HSA Port Connections required for the Formation Acquisition Phase

Formation Acquisition Step	HSA Action	Related HSA Port Connection
GNC Performance Tests	GNC exchanges state information with other spacecraft	GNC Controller → CCSDS Framer → IPC Driver
GNC Performance Tests	GNC executes propulsive maneuvers	GNC Controller → CCSDS Framer
GNC Performance Tests	GNC Controller reads in temperature and pressure fields from PROP telemetry	GNC Controller → Poly Database
Fault Management Performance Tests	Fault Detection reads in telemetry from payload subsystems	IPC Driver → CCSDS Deframer → TLM Parser → Poly Database → Fault Detection
Fault Management Performance Tests	Fault Response receives fault diagnosis and chooses the most suitable response	Fault Detection → Fault Response
Fault Management Performance Tests	Payload State Machine receives the recommended response and acts on it if allowed to do so	Fault Response → Payload State Machine
Enable Autonomy	Payload State Machine receives a ground command to enable autonomous actions	IPC Driver → CCSDS Deframer → CMD Dispatcher → Payload State Machine

Science Campaign

The science campaign phase of the mission is the most complicated portion of the mission due to the number of coupled actions between the GNC, ADCS, and science instruments. The timeline of tasks that the HSA accomplishes during this phase is shown in Fig 11 and Fig 12. These figures are adapted from the work first done in Ref 2. First, the ground verifies the feasibility of the science campaign by assessing the health of the payload subsystems on both spacecraft as well as characterizing the amount of delta-v and science data storage that is available on board. After confirming that a science campaign is possible, ground operators configure relevant science parameters such as the observation target on the sun, number of observation frames, exposure time, and parameters for image compression algorithms. The last ground-based command that is sent is a command to the PSM to set the current mode to Transfer mode.

At this point, the ground-based setup of the science campaign is complete, and the spacecraft takes over to autonomously complete the science campaign. The first task the PSM does is switch the primary spacecraft pointing constraint to be GPS-to-Zenith so that the GNC algorithms can receive quality navigation data for the initialization of its navigation algorithm.[2] After the navigation algorithm finishes its initialization sequence, the algorithm plans a set of maneuvers that will take the formation from the standby configuration to the science configuration.

Once the spacecraft is in the correct relative orbit configuration for Science mode, the PSM switches its internal mission mode to Science. Once this occurs, the CSIE gets turned ON and is passed the aforementioned science parameters. The GNC algorithms plan and execute additional propulsive maneuvers to drive the formation to the observation configuration when the spacecraft are over either of the Earth's poles. Immediately before the science observation, the HSA executes a series of commands to point the spacecraft in the correct target direction, turn OFF the magnetorquers and UHF downlink, and turn ON the Laser Range Finder. The magnetorquers and UHF downlink are turned OFF to ensure that there is no electromagnetic interference with the science instrumentation during a science observation. The port connections required for this phase of the mission are documented in Table 13.

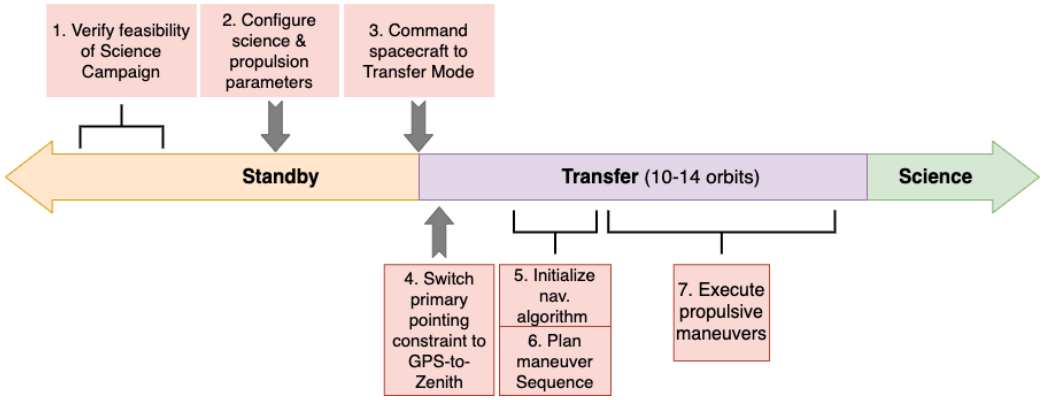

Figure 11. Initiation of Science Campaign

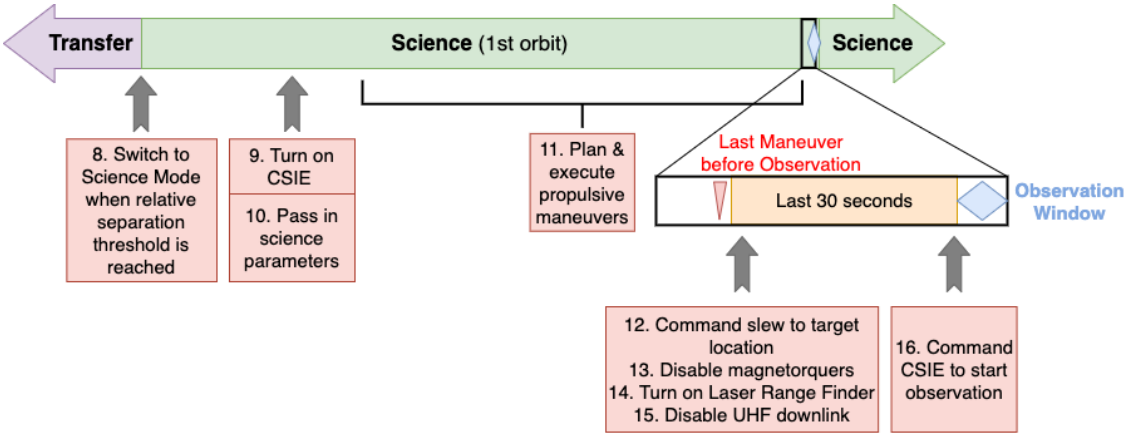

Figure 12. Science Orbit and Observation Timeline

Table 13. HSA Port Connections required for the Science Campaign Phase

Science Campaign Step	HSA Action	Related HSA Port Connection
Configure science parameters	HSA accepts a command to set desired target observation location	IPC Driver → CCSDS Deframer → CMD Dispatcher → Payload State Machine
Configure science parameters	HSA sends desired observation location to GNC Controller for their use in algorithms	Payload State Machine → GNC Controller
Command Transfer Mode	PSM accepts a command to switch mission modes	IPC Driver → CCSDS Deframer → CMD Dispatcher → Payload State Machine
Switch Pointing Constraint	PSM autonomously commands BCT to perform a switch in the pointing constraint	Payload State Machine → CCSDS Framer → IPC Driver
Execute Propulsive Maneuvers	GNC Controller commands propulsive maneuvers	GNC Controller → CCSDS Framer
Disable/Enable Subsystems	PSM enables or disables subsystems based off of the	Payload State Machine → CCSDS Framer → IPC Driver
Command an Observation	GNC Controller will alert the HSA when an observation needs to occur	GNC Controller → Payload State Machine → CCSDS Framer → IPC Driver

Off-Nominal Scenarios

The final mission phase relevant to the HSA is unique in that it does not have a predetermined linear timeline of events. Instead, this phase has multiple different entry criteria that each illicit a different response from the spacecraft formation. However, from a software perspective, complexity is minimized by having the same set of Fprime components handle all fault conditions. This ensures that the data flow between components is the same regardless of what the actual mission scenario is. The fault management system must ensure that the fault diagnosis between the spacecraft is identical. As a result, the fault detection component must be continuously exchanging its diagnosis information as shown in Fig 13.

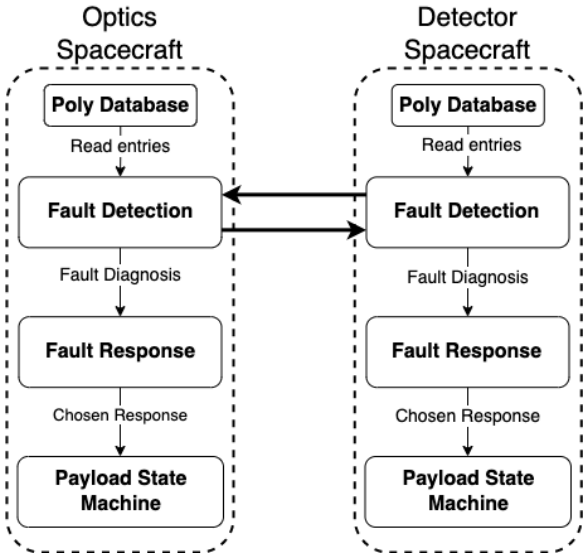

Figure 13. Distributed Fault Management System Data Flow

The fault detection data flow is enabled by the continuous stream of incoming packets from each of the payload subsystems. Since each packet is different, the TLM Parser component must identify the packet by its APID and correspondingly unpack the message. Once the packet is unpacked into its respective telemetry fields, the TLM Parser component will write the values into the Poly Database. From here, the fault detection component queries items in the database and diagnoses the fault, if any. Regardless of if there is a fault occurring, the fault detection component will exchange diagnosis information with the other spacecraft. However, if there is a fault, the FD component sends the diagnosis to the FR component which determines the appropriate fault response for the specific scenario.[14] Finally, the FR component sends its response recommendation to the PSM which actuates the response. If autonomy is enabled by the ground, the PSM will update its internal variables (mission mode, subsystem states, formation role) according to the response recommendation and alert the GNC Controller and/or the PSM on the other spacecraft of any changes as necessary. The port connections to enable these interactions is shown in Table 14.

Table 14. HSA Port connections required for off-nominal scenarios

Off-Nominal Scenarios Step	HSA Action	Related HSA Port Connection
Receive Payload Telemetry	Poly DB updates the database with the received subsystem telemetry	IPC Driver → CCSDS Deframer → TLM Parser → Poly Database

Read Telemetry for Fault Detection	Fault Detection polls the telemetry database to retrieve the telemetry values it is interested in	Fault Detection → Poly Database
Determine Fault Diagnosis	Fault Detection passes diagnosis information to the Fault Response component	Fault Detection → Fault Response
Fault Response Recommendation	Fault Response passes the PSM its recommendation for corrective action	Fault Response → Payload State Machine
Payload State Machine actuates fault response	PSM alerts other spacecraft of mode or role switch change	Payload State Machine → CCSDS Framer → IPC Driver
Payload State Machine actuates fault response	PSM receives an alert from other spacecraft of mode or role switch change	IPC Driver → CCSDS Deframer → Payload State Machine
Payload State Machine actuates fault response	PSM alerts the GNC Controller of mode or role switch change	Payload State Machine → GNC Controller

HSA IMPLEMENTATION AND TESTING

With the HSA topology defined, the next step is to begin the implementation phase of the flight software. A high level overview of this phase is given in Fig 14. The Fprime framework uses a domain specific modeling language called FPP (F Prime Prime) to configure the interfaces of a component and topology. The first step to defining a component starts with defining the data types of each of its input and output ports in FPP. These ports can then be used in the FPP file of the component itself. After all custom and built-in ports are defined, other commands, telemetry, and parameters can be included in the component FPP file. Once the component definition is complete, the initial boilerplate C++ code is autogenerated through Fprime based on the FPP definitions.[22] This gives users a starting point for developing the logic of their Fprime component.

After all the work to define the scope of the components, write requirements, and generate the template C++ structure, the logic of the component can finally be written. At this point, with all the scaffolding in place, actually developing the software is less cumbersome as the developer only has to be worried about the functionality of one component at a time. However, software development is an iterative process, so the implementation of the component will be continuously updated as it goes through its test campaign. Just as with any other software system, modules of code in Fprime must undergo both unit testing and integration testing. To complete these tests, Fprime provides a large suite of development tools.

Unit testing in Fprime begins in a similar fashion to component development – the boilerplate template for tester code is generated based on the definitions of the component FPP. The autogenerated tester code provides a harness to easily test the input and output functions of each component. The tester component has complete access to the states and variables of the component C++ code and as a result, can be written to completely validate all functionalities of the component.[21] This allows for black-box and white-box testing, depending on the flight software's testing requirements. By setting up the framework in this manner, every Fprime component can be unit tested in isolation without any other components. The extent to which a unit test verifies the functionality of the component can often be characterized by the code coverage of the unit test. Code coverage gives qualitative results on how many lines and logical branches of the components C++ code were tested via the unit tests. For the VISORS mission, the goal for each component's unit test code coverage was 80%. Note that a high code coverage metric does not always mean that the component is working nominally. There are often cases where a component cannot be fully tested unless it is interfaced with other components in the topology. This leads to the next phase of the software testing framework – Integration Tests.

For the VISORS mission, integration testing was completed in two main ways. First, multiple related components were tested together through an exerciser deployment. This test deployment is different from the

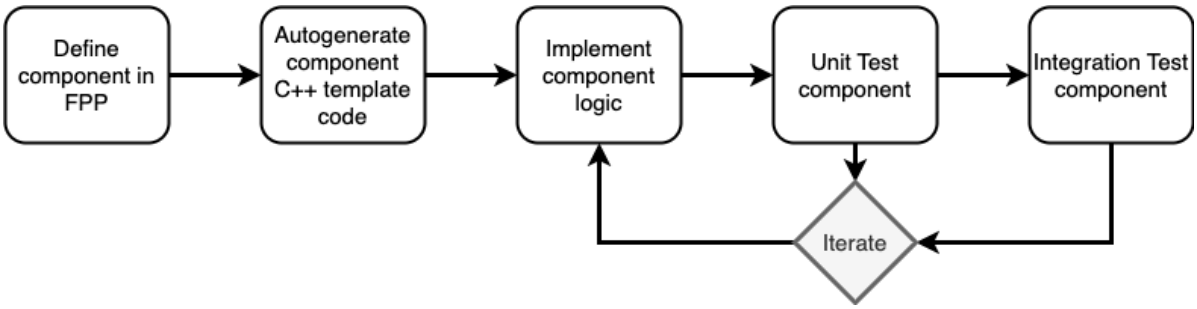

Figure 14. Fprime Component Level Implementation and Testing Flowgraph

flight deployment in that it is specifically tailored to exercise a small subset of components working together. By separating the flight deployment into smaller sub-deployments, the functionality and operation of the components can be more easily debugged and verified since fewer overall tasks are happening in the sub-deployment. Almost all the custom components written for the VISORS mission are included in at least one exerciser deployment. Exerciser deployments are almost exclusively run on a desktop computer instead of the flight hardware and may use the Fprime Ground Data Segment (GDS) instead of the ground software used during flight. This allows for faster and more iterative development. Tests using the Fprime GDS can be manually implemented or scripted in Python using the GDS Application Programming Interface (API), giving users the option to completely automate their integration tests.

However, to verify that the component also works while running on a hardware platform, hardware integration tests are developed. The Fprime deployments used for these tests are unique in that they require the components that enable communication between the hardware device and the test computer. For VISORS, hardware integration tests must be compatible with the COSMOS ground operations software. Users can use the Command and Telemetry Server in Cosmos to manually send commands to any subsystem or automate the sending of commands with scripts. It is important to note that scripts in COSMOS are written using the Ruby programming language, not Python.[23] Thus, any scripts written in Python for integration testing with the Fprime GDS must be adapted to work with COSMOS instead.

After each of the custom Fprime components for VISORS go through this implementation and testing phase, the components are integrated into flight deployments. Once the deployment includes the components planned for that specific flight software release, the flight deployment undergoes system-level integration testing where it runs on the testbed or flight hardware and interfaces with all payload subsystems. This helps verify the data interfaces between subsystems and ensures that the entire spacecraft system operates nominally. Due to the fact that not all payload functionality (such as propulsive maneuvering) can be tested on the ground, the system level integration tests may need to incorporate spoofed or simulated data from subsystems.[6]

CONCLUSION

As formation flying missions become more abundant, multi-functional software systems like the HSA will become more common. As the complexity of these software systems increase, it is vital to carefully consider how these systems should be architected so that time can be saved on the development of the system. The contents of this paper aimed to show this in action by providing a concise look at the design and development process of the HSA flight software deployment for the VISORS mission. By thinking critically about what drives the design of the HSA, a robust, modular architecture was developed. The choice of the Fprime software framework facilitated the design of the software system and enabled fast, concurrent development among all team members. This concurrency allowed for more time for testing and provided a higher level of confidence that this ambitious mission could be successful regardless of schedule, cost, and staffing constraints. As the development of the HSA continues into the future, developers should work with mission operators to further iterate the design within the confines of the mission requirements. Further iteration will result in a system that is even more user-friendly while also increasing reliability and robustness for ground based testing and on orbit operations.

ACKNOWLEDGMENTS

The work to develop the HSA for the VISORS mission is funded by the National Science Foundation under grant No. 1936576. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

1. Kulu, “E. Nanosatellite Launch Forecasts 2022 - Track Record and Latest Prediction.” Small Satellite Conference 2022.
2. Lightsey, E. G., Arunkumar, E., Kimmel, E., Kolhof, M., Paletta, A., Rawson, W., Selvamurugan, S., Sample, J., Guffanti, T., Bell, T., Koenig, A., Amico, S. D. ’, Park, H., Rabin, D., Daw, A., Chamberlin, P., and Kamalabadi, F. “Concept of Operations for the VISORS Mission: A Two Satellite CubeSat Formation Flying Telescope”. 44th Annual AAS Guidance, Navigation & Control Conference, Breckenridge, CO, 4-9 February 2022.
3. Koenig, A. W., D’amico, S., and Lightsey, E. G. “Formation Flying Orbit and Control Concept for the Visors Mission”. AIAA SciTech 2021 Forum, Virtual Event, 11-15 & 19-21 January 2021.
4. Hart, S. T., Daniel, N. L., Hartigan, M. C., and Glenn Lightsey, E. “Design of the 3-D Printed Cold Gas Propulsion Systems for the VISORS Mission”. 44th Annual AAS Guidance, Navigation & Control Conference, Breckenridge, CO, 4-9 February 2022.

5. Stanford University. “Formation Alignment During Observation, VISORS Systems Integration Review”. 2023.
6. Kimmel, E., Paletta, A., Arunkumar, E., Krahn, G., and Glenn Lightsey, E. “Testing methodology for Spacecraft Precision Formation Flying Missions”. 45th Annual AAS Guidance, Navigation & Control Conference, Breckenridge, CO, 4-8 February 2023.
7. Baron, D. “Image of DSC During Testing at BCT.” 2023.
8. Schulte, P. Z. “A State Machine Architecture for Aerospace Vehicle Fault Protection”. 2018.
9. Payne, J. “VISORS XB1 Spacecraft Bus Interface Control Document”. 2022.
10. R. K. Kandt, "Experiences in Improving Flight Software Development Processes". IEEE Software, vol. 26, no. 3, pp. 58-64, May-June 2009
11. Jones, M., Fretz, K., Kubota, S., and Smith, C. A. “The Use of the Expanded FMEA in Spacecraft Fault Management.” Annual Reliability and Maintainability Symposium (RAMS, 2018.
12. McDonald, D., and Ligthtsey, E. G. “Fault Management in Small Satellites Lessons Learned from the Lunar Flashlight and ARMADILLO Missions.”
13. Paletta, A., Lightsey, G., Rawson, W., Arunkumar, E., Kimmel, E., Selvamurugan, S., Hauge, M., and Guffanti, T. “Development of a Contingency Operations Architecture for the VISORS Formation Flying Space Telescope.” 2022.
14. Paletta, A., and Lightsey, G. “Development of an Autonomous Distributed Fault Management Architecture for the VISORS Mission.” 2023.
15. “Recommendation for Space Data System Standards BLUE BOOK RECOMMENDED STANDARD SPACE PACKET PROTOCOL.” 2020.
16. Krishnaveni, M. S., and Ruby, M. D. “Comparing and Evaluating the Performance of Inter Process Communication Models in Linux Environment.” 2016.
17. Tarr, P., Ossher, H., Harrison, W., and Sutton, S. M. N Degrees of Separation: Multi-Dimensional Separation of Concerns. 1999.
18. The F′ Framework Team. F′: A Flight-Proven, Multi-Platform, Open-Source Flight Software Framework.
19. Mccomas, D., Wilmot, J., and Cudmore, A. “The Core Flight System (CFS) Community: Providing Low Cost Solutions for Small Spacecraft.” 2016.
20. Bocchino, R. L., Canham, T., Watney, G. J., Reder, L. J., and Levison, J. “FPrime: An Open-Source Framework for Small-Scale Flight Software Systems.” Small Satellite Conference 2018.
21. Rizvi, A., Ortega, K. F., and He, Y. “Developing Lunar Flashlight and Near-Earth Asteroid Scout Flight Software Concurrently Using Open-Source F Prime Flight Software Framework.” Small Satellite Conference 2022.
22. Bocchino, R. L., Levison, J. W., and Starch, M. D. “FPP: A Modeling Language for F Prime”. IEEE Aerospace Conference 2022.
23. Melton, R. “Ball Aerospace COSMOS Open Source Command and Control System.”