CubeSat Reusable Interface Software Platform (CRISP): A Lightweight Message-Bus-Based Flight Software Architecture for Rapid Payload Integration

Rory H. Scobie[1], Kevin D. Kaufeld[1], Bradley J. Hoose[1], Keith S. Morgan[1], Kimberly K. Katko[2], Markus P. Hehlen[2], John M. Michel[1]
Los Alamos National Laboratory, [1] Space Data Science & Systems (ISR-3), [2] Space & Remote Sensing (ISR-2)
Los Alamos, NM 87545; 505 606 2054
rscobie@lanl.gov

ABSTRACT

The Agile Space portfolio of projects at Los Alamos National Laboratory (LANL) develops low-cost, rapidly-deployable space payloads and systems. To increase the agility of future missions, we are developing CRISP: the CubeSat Reusable Interface Software Platform. CRISP provides a lightweight and reusable flight software framework for rapid integration of custom payloads with commercial microsatellite platforms. CRISP cuts development time and costs by reducing non-recurring engineering (NRE); thereby accelerating mission agility. To achieve these goals, CRISP provides a core set of payload/data management functions and abstracts the interface between the bus avionics and the payload(s). CRISP currently consists of the following core software modules: a lightweight and scalable publish-subscribe message bus, a space vehicle interface, volatile and nonvolatile memory management, time and ephemeris distribution, debug printing and logging, and watchdogs. We have also developed a modular ground support utility to ease integration and testing, as well as a template flight software application that can be quickly adapted to new missions. Two upcoming CubeSat missions at LANL have already adopted CRISP: the Experiment for Space Radiation Analysis (ESRA) and the Mini Astrophysical MeV Background Observatory (MAMBO).

BACKGROUND

Traditional space missions are typically characterized by a low risk tolerance and the associated high cost and long schedules. The Agile Space portfolio of projects at Los Alamos National Laboratory (LANL) aims to create a capability that enables higher-risk missions to be executed at lower cost and on shorter timelines. This opens the door to a range of new missions for science, technology demonstration, and constellations. Additionally, increased agility allows for faster response to emerging national and global security threats.[1]

A core element of LANL's approach to Agile Space is a mission-agnostic technical and logistical framework that leverages commercial technology and establishes standardized hardware, software, and workflows (Figure 1). This reduces Non-Recurring Engineering (NRE), a major cost and schedule driver, and builds flight heritage. As a result, the primary mission risk is properly focused on the typically novel state-of-the-art payload.

One key component of this approach is CRISP: the Cubsat Reusable Interface Software Platform. CRISP enables the payload processor to become a standardized, reusable, and highly-functional broker between the mission-specific payload and the commercial host bus avionics (Figure 2).

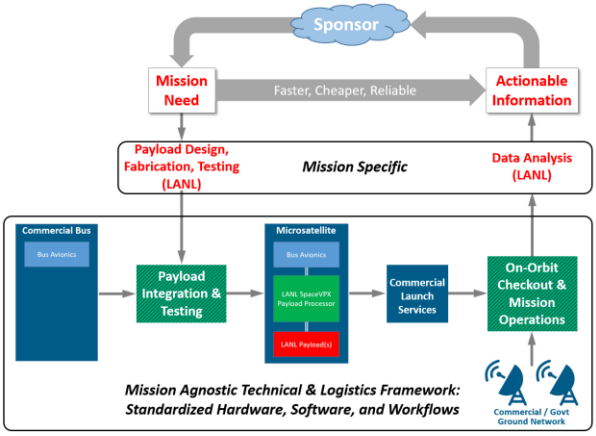

Figure 1: Envisioned LANL Agile Space capability that revolutionizes the time and resources needed to produce actionable information for a given mission need. Mission-specific aspects (red), commercial technology (blue), and standardized workflows (green) are shown.

We evaluated a variety of existing small-satellite software platforms that could potentially offer the desired functionality. NASA's Core Flight System (cFS)[2] framework and Xplore's KubOS[3] both employ a publish-subscribe architecture; however, they both impose a significant level of overhead that was not compatible with our need for a low-resource lightweight

implementation. Likewise, generic publish-subscribe frameworks such as ZeroMQ[4], ZCM[5], MQTT[6], and ROS[7] were considered and dismissed for similar reasons.

While these existing frameworks are excellent pieces of software, the need for a minimalist, lightweight, scalable, and easily reusable implementation warranted the development of the new publish-subscribe architecture of CRISP.

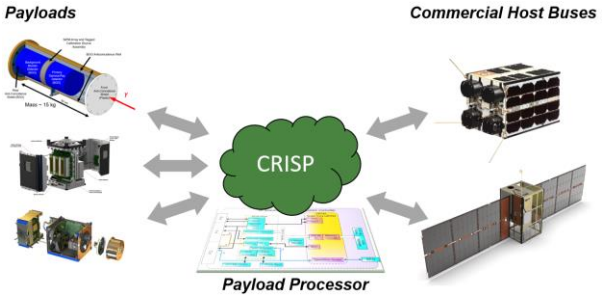

Figure 2: CRISP runs on the payload processor and brokers the interface between the mission-specific payload and the standard commercial host bus.

DESIGN

Overview

Figure 1 shows a top-level architecture of CRISP. CRISP employs a publish-subscribe architecture to divide functionality into multiple submodules whose communication is coordinated by a central broker. CRISP includes a collection of submodules to provide functionality commonly used in payload flight software, such as data storage and State of Health (SOH) collection. In addition, mission-specific submodules can be easily created using a standard interface with the message bus. In general, the message bus and the submodules are highly configurable and easily modifiable in order to adapt to a wide variety of payload requirements.

CRISP Message Bus

The CRISP Message Bus is the foundation of any CRISP-based project and is responsible for data transfer between submodules. It acts as a broker between publisher submodules and subscriber submodules, feeding published messages to their intended subscriber based on the message type.

Figure 4 shows the architecture of the message bus. CRISP submodules can register with the broker as a publisher, subscriber, or both. Publishers write messages to a single POSIX message queue, which the broker processes continuously in a POSIX thread.

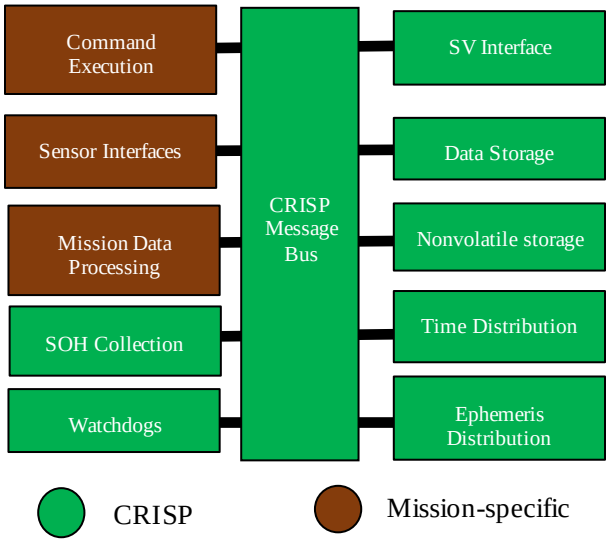

Figure 1: Top-level architecture of a CRISP-based project.

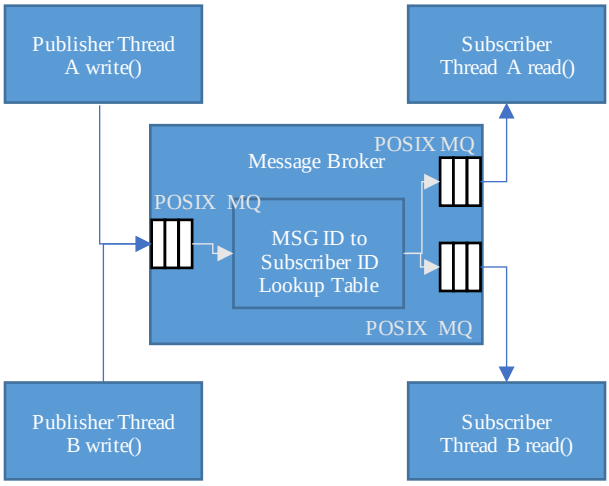

Figure 2: Message Bus Design, arrows denote data flow.

All messages published on the message bus start with a header, as described in Table 1. In order to receive messages, subscribers provide the broker with a map of message types (as indicated in the message type field of the header) and function pointers. The broker keeps track of these mappings in a lookup table, and also maintains a POSIX message queue for each subscriber. When the broker reads a message from its publish queue, it writes this message to the queue of each subscriber to this message type.

When subscribers call the message bus read function they have the option of blocking until a message arrives or returning immediately. When a message is available,

the read function invokes the callback previously registered for the message.

Table 1: Message Bus Header Format

Field Name	Field Type	Purpose
Message Marker	uint16	Maury & Styles frame synchronization code[8], used to identify CRISP messages in memory.
Message Type	uint16	Used by broker to forward messages to the correct subscriber and message handler
Sequence Number	uint32	Used to ensure messages are processed in the correct order

Space Vehicle Interface

The Space Vehicle (SV) Interface provides a standard interface for the payload to communicate with the space vehicle, regardless of the specific space vehicle used. The SV-specific implementation of this interface is modularized within the SV interface, so that new space vehicles can be easily added without heavily modifying the code base (Figure 3).

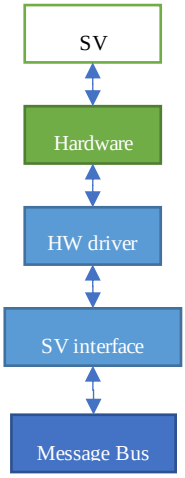

Figure 3: Space Vehicle Interface Design.

This standard interface allows the payload to receive commands from the space vehicle and ground, as well as send telemetry back. Additionally, this interface allows for time and ephemeris information to be received from the space vehicle. The space vehicle interface has two modes: It can stream data to the SV for immediate downlink, or it can send data to the SV for temporary storage.

Storage Manager

The storage manager is responsible for storing data in memory for later transmission to the ground. Additionally, it is responsible for chunking messages to fit the Maximum Transmission Unit (MTU) of the SV interface and transmission to the SV interface. Figure 4 shows a diagram of the storage manager architecture .

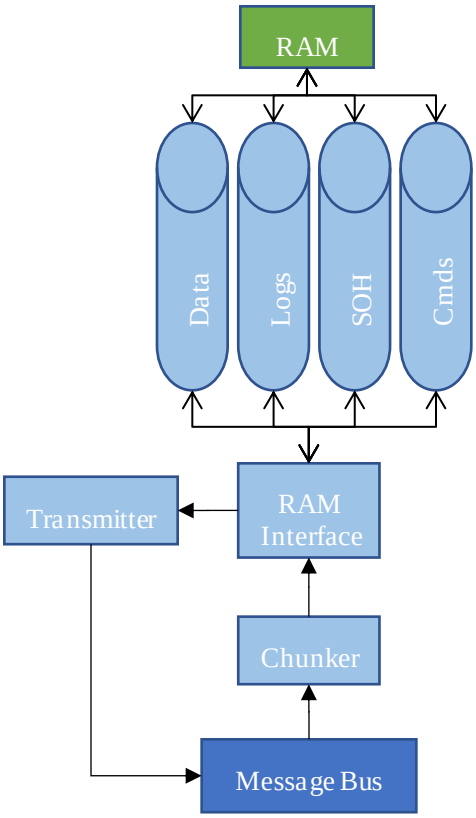

Figure 4: Storage Manager Design. Green denotes hardware, dark blue denotes another related submodule, light blue denotes Storage Manager.

The storage manager keeps track of science data, state of health, log messages, and the recent command history. Each of these data types is given a dedicated ring buffer, and data written to a buffer is stored sequentially and without padding in order to save space. Pointers and counters are used to keep track of buffer state, and these are stored in nonvolatile memory in case of a reboot. When the ground needs to retrieve data, a message is sent to request a certain number of bytes. Starting from the pointer to the last untransmitted byte in the buffer, the data is divided into chunks sized at the MTU of the SV interface. If the last chunk is smaller than this, a partial chunk is sent.

Nonvolatile Memory Manager

The nonvolatile memory (NVM) manager is responsible for all interactions with protected regions of nonvolatile memory. This includes interacting with the Memory Management Unit (MMU) to lock/unlock regions of memory, write data, and handle new software uploads

and parameter changes. Figure 5 gives an overview of the NVM Manager architecture.

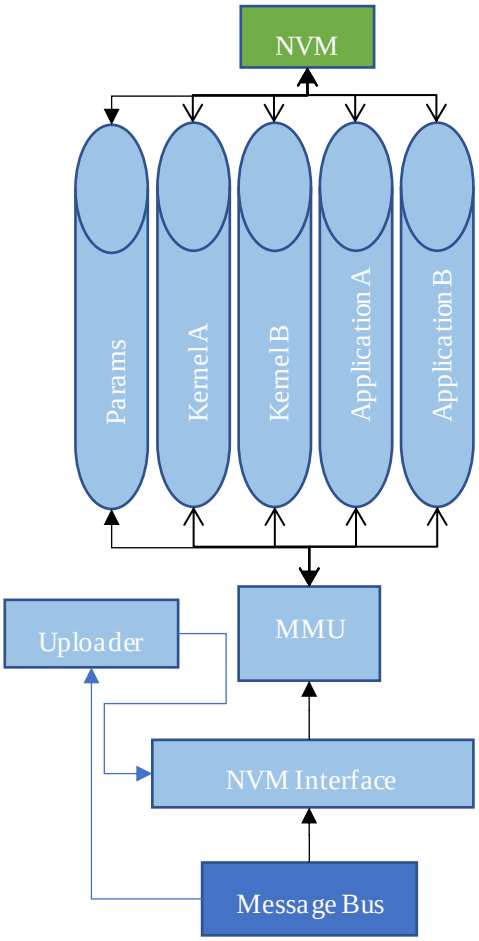

Figure 5: Nonvolatile Memory Manager Design.

To support a wide variety of payload processing platforms, the NVM Manager abstracts interaction with the MMU behind a standard interface, with the platform-specific MMU-related functions registered to the NVM Manager during initialization of the submodule. This allows CRISP-based applications to be ported to new hardware with minimal NRE.

Additionally, the NVM Manager is responsible for handling software uploads and parameter changes from the ground. This can be to either a specified memory bank or parameter if these are defined in the configuration, or to an arbitrary address. The ground segment divides upload data into chunks based on the SV Interface MTU and appends additional data and a CRISP message bus header. The integrity of each chunk is verified using a Cyclic Redundancy Check (CRC), as is its continuity using the sequence field in the header. Data is stored in a buffer until all integrity checks pass, at which point the ground sends a final confirmation to commit the data to nonvolatile memory. This upload process is outlined in Figure 6. CRISP's Ground Support Equipment (GSE) includes the capability to upload binary data to the payload using this scheme.

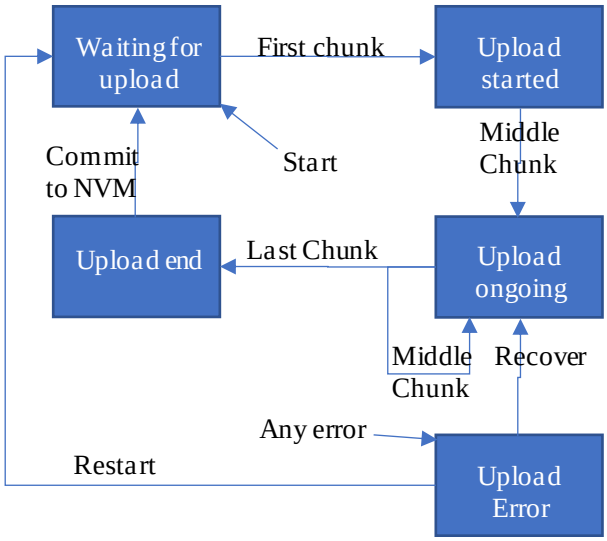

Figure 6: CRISP Upload Process.

Other Submodules

CRISP provides a number of utilities and support submodules for convenience as well as code reuse among submodules. These are summarized in Table 2

Table 2: Other Submodules

Name	Description
Ephemeris Manager	Periodically collects ephemeris data from the SV interface and distributes it to submodules that subscribe to ephemeris telemetry message types
Time Manager	Periodically collects time data from the SV in order to prevent clock drift.
State of Health Manager	Periodically collects SOH data from all submodules and interfaces with storage manager for storing in memory for later downlink
Print Manager	Assists in the logging of messages for debugging and error reporting, both in memory for later downlink and on the command line. Allows for filtering by severity level.
Watchdogs	CRISP provides various watchdogs to reboot submodules in case of an unrecoverable software error.
Command Line Utilities	CRISP provides various command line utilities to aid in development.

Ground Support Equipment

CRISP provides a basic Ground Support Equipment (GSE) utility to interface with CRISP-based payloads. This GSE is able to inject a list of messages with optional delays between injections into the CRISP Message Bus via the SV Interface. Since commands and their structure

will vary from mission to mission, the GSE ingests a dictionary of commands and their structure at runtime. Processing is automatically done to ensure compatibility with the CRISP message bus. Telemetry output from the payload via the SV interface is captured and written to a binary file for later processing. In addition, the GSE is capable of uploading data to the payload via the NVM Manager's upload capability. Figure 7 shows the architecture of the GSE.

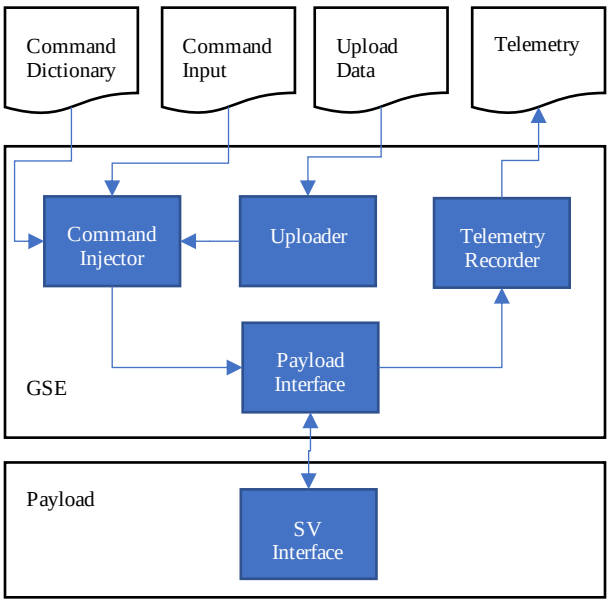

Figure 7: Ground Support Equipment Design. Arrows indicate data flow.

Template Application

A template application has been developed to serve as a starting point for new CRISP-based Flight Software. This application includes all CRISP submodules, the CRISP Message Bus, and an RTEMS-based shell as a placeholder application.

ADOPTION AND INTEGRATION

CRISP has been adopted as the payload flight software platform for several cutting-edge space radiation research endeavors underway at LANL, including MAMBO and ESRA. These payloads will be integrated with the Nanoavionics M12P, a commercial 12U CubeSat bus. Additionally, both use LEON-based processor cards designed by LANL for payload processing[9, 10]. RTEMS has been chosen as the payload processor operating system due to its support on LEON processors. Through the use of CRISP, both of these projects have already seen a significant reduction in NRE during the development of flight software.

MAMBO

The Mini Astrophysical MeV Background Observatory (MAMBO) is an upcoming CubeSat mission in gamma-ray astronomy. Its mission is to make high-quality measurements of cosmic diffuse gamma-ray (CDG) background in the 0.3–10 MeV energy range. Due to its innovative shielded spectrometer design and relatively low instrument background afforded by its small size, MAMBO will provide the best measurements ever made of the MeV CDG spectrum.[11]

ESRA

The Experiment for Space Radiation Analysis (ESRA) is an upcoming CubeSat mission under development at LANL. Its mission is to aid in the development of the next generation of plasma and energetic charged-particle sensors.[12]

Integration Requirements and Procedure

In order to integrate CRISP into a payload's flight software, the flight software must run on a POSIX-compliant operating system. CRISP has been tested with RTEMS and Linux, but in theory any POSIX-compliant OS should be supported.

Some configuration of each of CRISP's submodules may need to occur based on the memory and speed constraints of the payload processor as well as application-specific requirements. Each submodule, including the message bus, includes a variety of configuration parameters and flags in order to tune CRISP and enable/disable features as necessary.

Though CRISP supports a limited number of hardware components, the software architectures of the submodules that interface with hardware decouple the specifics from the rest of the submodule. Thus, if an application requires adding support for a new SV, MMU, or nonvolatile memory, hardware support can be added with minimal or no change to the submodule's application-level code.

Finally, mission-specific submodules must be added to handle the interfaces with instruments or other hardware, the execution of mission-specific commands, the processing of mission data, and any other mission-specific behavior. Due to CRISP's publish-subscribe architecture, a mission-specific submodule for an instrument typically publishes science data and its state of health, subscribes to relevant commands from the ground and telemetry from other submodules, and executes any periodic logic within its own thread.

ONGOING AND FUTURE WORK

CRISP currently supports one space vehicle (NanoAvionics M12P), two payload processors (LEON3 and LEON4), and two operating systems (RTEMS and LINUX). The need to deliver to missions that have adopted CRISP prioritize the support of the Space Vehicles, Payload Processors, and Operating Systems used by these missions. However, as new missions adopt CRISP, support for more of these will be added as needed.

ACKNOWLEDGMENTS

Research presented in this **PAPER** was supported by the Laboratory Directed Research and Development program of Los Alamos National Laboratory under project number 20210701DI.

REFERENCES

1. Dallmann, N., Delapp, J., Enemark, D., Fairbanks, T., Fortgang, C., Guenther, D., Judd, Stephen., Kestell, G., Lake, J., Prichard, D., Proicou, M., Quinn, H., Reid, R., Schaller, E., Seitz, D., Stein, P., Storms, S., Sullivan, E., Tripp, J., Warniment, A., Wheat, R., "An Agile Space Paradigm and the Prometheus CubeSat System," 29th Annual AIAA/USU Conference on Small Satellites, Logan, Utah (August 2015)
2. NASA, "Core Flight System" (2020) [source code] "https://github.com/nasa/cFS
3. Xplore Inc., "KubOS" (2020) [source code] https://github.com/kubos/kubos
4. The ZeroMQ Authors, "The ZeroMQ project" (2022) [source code] https://github.com/zeromq
5. The ZeroCM Authors, "ZCM: Zero Communications and Marshalling" (2021) [source code] https://github.com/ZeroCM/zcm
6. OASIS, "MQTT Version 5.0," (March 2019) [website] https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html
7. Open Robotics, "Robot Operating System," (2021) [source code] https://github.com/ros/ros
8. Maury, J., Styles, F., "Development of Optimum Frame Synchronization Codes for Goddard Space Flight Center PCM Telemetry Standards," 1964 National Telemetering Conference, Los Angeles, California (June 1964)
9. Merl, R., Cox, E., Dutch, R., Graham, P., Larsen, S., Michel, J., Milby, D., Morgan, K., and Tripp, K., "LEON4 Based Radiation-Hardened SpaceVPX System Controller," 2020 IEEE Aerospace Conference, Big Sky, Montana (March 2020)
10. Wiens, R. et. al., "The SuperCam Instrument Suite on the NASA Mars 2020 Rover: Body Unit and Combined System Tests," figure 24, Space Sci Rev 217, 4 (2021)
11. Bloser, P., Vestrand, T., Hehlen, M., Parker, L., Beckman, D., McGlown, J., Holguin, L., Katko, K., Sedillo, J., Nelson, A., Lee, G., "The Mini Astrophysical MeV Background Observatory (MAMBO) CubeSat Mission," 35th Annual Small Satellite Conference, Logan, Utah (August 2021)
12. Maldonado, C. A. et al., "The Experiment for Space Radiation Analysis: A 12U CubeSat to Explore the Earth's Radiation Belts," IEEE Aerospace Conference, Big Sky, Montana (2022)