The Troupe System: An Autonomous Multi-Agent Rover Swarm*

Nathaniel A. Benz†, Irfan Sljivo‡, Aaron Woodard§, Pavlo G. Vlastos¶, Corey K. Carter‖, and Mohammad Hejase**
Intelligent Systems Division, NASA Ames Research Center, Moffet Field, CA, 94035.

Autonomous cooperative robotic systems are the future of space exploration. The complexity of such systems makes their development, verification and assurance challenging. The Robust Software Engineering group at NASA Ames has developed the Troupe project that aims to explore the design and development of a swarm of autonomous rovers tasked to perform autonomous exploration and mapping of an unknown terrain. In this paper, we showcase the system design, and accompanying verification and validation tools integrated in the Troupe system development life-cycle.

I. Introduction

The future of space exploration will leverage the power of multi-agent systems. Whether it is a satellite constellation in low-Earth orbit, or a swarm of zero-gravity construction drones, autonomous multi-agent systems provide the next step forward for carrying out large-scale space missions. Rover swarms in particular could embark on missions focusing on Lunar surface exploration. Swarms have the potential to yield high science utility returns. However, there are a number of design and implementation questions that have yet to be fully addressed.

The Troupe project serves to explore the design and development of multi-agent systems in the form of a rover swarm. The Troupe rover swarm will act as a case study for implementing research tools for safety risk management, requirement formalization, run-time verification frameworks, and other related verification tool-sets. The Robust Software Engineering (RSE) group at NASA Ames has two primary objectives: (1) research and development of tools to improve Verification and Validation (V&V) of safety-critical software and (2) design and deployment of flight software for small-sat spaceflight missions. Ideally, these two objectives will complement each other. The research teams develop tools for the mission developers to improve software quality and the mission developers provide direct feedback to the research team on use-cases and desired features. In practice, there have been limitations to the envisioned collaboration. Schedule constraints of a flight mission often do not allow for prototyping and training with tools in active development. Security constraints, such as ITAR data, prevent sharing of concrete use-cases.

To overcome these barriers, the RSE group has implemented an incubator program called Troupe which consists of four autonomous rovers that coordinate to map an unknown terrain. The final deliverable will be design, development and demonstration of the rovers mapping the rover-scape testing ground located at NASA Ames. The demonstration mission allows the development of spaceflight software while integrating advanced V&V tools including formal model-checkers, mathematical sound static analyzers, and run-time safety monitoring. While Troupe follows NASA's rigorous requirements for software development, the mission itself has no limitations with sharing data both with the tool developers and research community. This way the Troupe team can learn new tools that are in active development and provide feedback directly to the research tool developers.

A. NASA Ames RSE Tools

The RSE group at NASA Ames is currently using Troupe as a demonstration for existing verification and validation research tools (FRET, CoCoSim, Copilot, Ogma, AdvoCATE) and as a platform to develop and test new tools or research ideas. Each of these tools is described below.

*GOVERNMENT RIGHTS NOTICE. This work was authored by employees of KBR Wyle Services, LLC under Contract No. 80ARC020D0010 with the National Aeronautics and Space Administration. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, or allow others to do so, for United States Government purposes. All other rights are reserved by the copyright owner.

†Intelligent Systems Division, NASA Ames Research Center.
‡Intelligent Systems Division, NASA Ames Research Center/ KBR.
§Intelligent Systems Division, NASA Ames Research Center
¶Intelligent Systems Division, NASA Ames Research Center/ KBR.
‖Intelligent Systems Division, NASA Ames Research Center
**Intelligent Systems Division, NASA Ames Research Center

- FRET (Formal Requirement Elicitation Tool) [1, 2] is an open-source tool for writing, understanding, formalizing, and analyzing requirements. In practice, requirements are typically written in natural language, which is ambiguous and, consequently, not amenable to formal analysis. Since formal, mathematical notations are unintuitive, requirements in FRET are entered in a restricted natural language named FRETish [3] with precise unambiguous meaning. FRET helps users write FRETish requirements both by providing grammar information and examples during editing, but also through English and diagrammatic explanations to clarify subtle semantic issues. For each FRETish requirement, FRET generates formulae in a variety of formalisms including metric Linear Temporal Logic (LTL) and Lustre [4] code. Moreover, FRET supports interactive simulation of produced formalizations to ensure that they capture user intentions. Through its analysis portal, FRET connects to analysis tools by facilitating the mapping between requirements and models/code, and by generating verification code. Among other tools, FRET connects to: 1) the CoCoSim automated analysis tool for the verification of Simulink and Stateflow models [5]; and 2) the Copilot runtime monitoring tool for the analysis of C programs [6]. Finally, FRET also supports the consistency/realizability analysis of requirements, which can be used to identify conflicting requirements [7, 8]. Realizability is a notion of requirements consistency in the face of any expected input from the environment.
- CoCoSim (Contract-based Compositional verification of Simulink models) is an open-source plugin [9] for MATLAB Simulink/Stateflow, that allows users to formally verify requirements expressed in the form of Assume-Guarantee contracts [10, 11]. To tackle scalability issues, verification can be achieved using compositional reasoning over contracts defined for the system as a whole, as well as its subsystems. Complexity induced by verbose subsystems can be abstracted away, by replacing the model with its corresponding contract in the proof. Besides compositional verification, CoCoSim provides means to translate Simulink/Stateflow models into equivalent implementations in Lustre and C, and limited support for test-case generation.
- IKOS (Inference Kernel for Open Static Analyzers) [12] is an open-source static analyzer developed at NASA Ames for C/C++ based on the theory of Abstract Interpretation [13]. It can detect or prove the absence of runtime errors (e.g, buffer overflows, integer overflows, null pointer dereferences, etc.) in the source code. IKOS uses Abstract Interpretation techniques to compute an over-approximation of all the reachable states of the program. IKOS is currently being used in the Troupe project for static analysis of the developed cFS apps.
- Copilot [14, 15], is an open-source runtime verification framework for real-time embedded systems. The framework translates monitor specifications into C99 code with no dynamic memory allocation that executes with predictable memory and time, crucial in resource-constrained environments, embedded systems, and safety-critical systems. The language uses advanced programming features to provide additional compile-time and runtime guarantees. For example, all arrays in Copilot have fixed length, which makes it possible for the system to detect, before the mission, some array accesses that would be out of bounds. Copilot supports a number of logical formalisms for writing specifications including Bounded Linear Temporal Logic (BLTL), Past-time Linear Temporal Logic (PTLTL), and Metric Temporal Logic (MTL), and includes libraries with functions such as majority vote, used to implement fault-tolerant monitors [16]. Copilot follows NASA NPR7150.2C (for Class D software).
- Ogma [17] is a tool to generate runtime monitoring applications. Ogma translates the input from high-level languages, such as the temporal logic formulas produced by FRET, and produces full runtime monitoring applications. Among others, Ogma can produce applications suitable to run in NASA's Core Flight System (cFS), ROS 2, and F'. The applications produced by Ogma automatically subscribe to the messages (in cFS) or topics (in ROS 2) in the software bus that provide the information that has been mentioned in the properties or requirements to monitor. That makes Ogma able to generate applications that are ready to run and do not need to be modified prior to integration as part of a larger infrastructure. The core of the monitors generated by Ogma is implemented via Copilot, which guarantees that they will run in predictable memory and time. Together with the monitoring applications, Ogma also generates additional aids, such as logging applications (which print the results of the monitors to a log), and testing applications. Ogma follows NASA NPR7150.2D (for Class D software).
- AdvoCATE (Assurance Case Automation Toolset [18]) is a tool that supports the development and management of safety assurance cases. A safety assurance case comprises all the artifacts that are created during system development and verification that are needed to assure that the system is acceptably safe for its intended operation. Safety cases are often represented and documented in the form of a graphical argument that presents how the system safety goals have been achieved and supported by the various items of *evidence*, such as test results, simulations, and formal verifications. AdvoCATE supports a range of notations and modeling formalisms, including Goal Structuring Notation (GSN) [19] to document safety cases and Bow-Tie Diagrams [20] for risk modelling. Some of the artifacts can be created directly in AdvoCATE (e.g., hazard log, safety arguments), while other artifacts

such as formal verification results, can be imported into the tool so that the evidence can be collectively viewed.

- R2U2 (Realizable, Responsive, and Unobtrusive Unit) [21–23] is an on-board monitoring system to continuously monitor system and safety properties of a cyber-physical system or its components. Health models within this framework [24] are defined using Metric Temporal Logic (MTL) and Mission-time Linear Temporal Logic (LTL) [22] for expressing temporal properties as well as Bayesian Networks (BN) for probabilistic and diagnostic reasoning. A signal processing unit reads in continuous sensor signals or information from the prognostics unit and performs filtering and discretization operations.
- SafeMAP (Safe Multi Agent Planning) is a risk-aware framework that unifies disparate models for multi-agent systems in a Markovian process that allows for simultaneous system health monitoring, decision making under uncertainty, and multi-agent system collaboration [25]. SafeMAP generates mission plans in an interpretable fashion that allows for translation of search results into user-friendly mission plans that clearly outline and detail system dynamics evolution, system risk evolution, and system reward or objective evolution. The main innovation in the implementation of SafeMAP stems from the adaptation of tools and technologies utilized for Dynamic Probabilistic Risk Assessment tools into a tool used for risk-aware planning for multi-agent systems.

II. Overview of the Troupe Development Life-cycle

Fig. 1 shows the Troupe development and V&V life-cycle. We divided the life-cycle into three stages: concept, system and app development. In the concept stage, we define the mission objectives and specify the high-level requirements in two levels L1 and L2, where L1 represents the mission goals, and L2 the systems that will be developed to achieve those goals. For each system, we perform the system-level phase, where the system is further characterized, and more detailed L3 and L4 level requirements are developed. L3 requirements detail the different system functions and L4 subsystems achieving those functions. At this phase we develop the system design and architecture based on the specified requirements.

Since the cFS architecture is app-oriented, we specify the app-level L5 requirements, which are implementation specific, and allocate them to the respective cFS apps. Since there are different kinds of apps in the system, the App development phase differs based on the type of the app. For example, traditional apps are developed manually, while for the Model-based apps and the Monitoring apps, the source code is automatically generated.

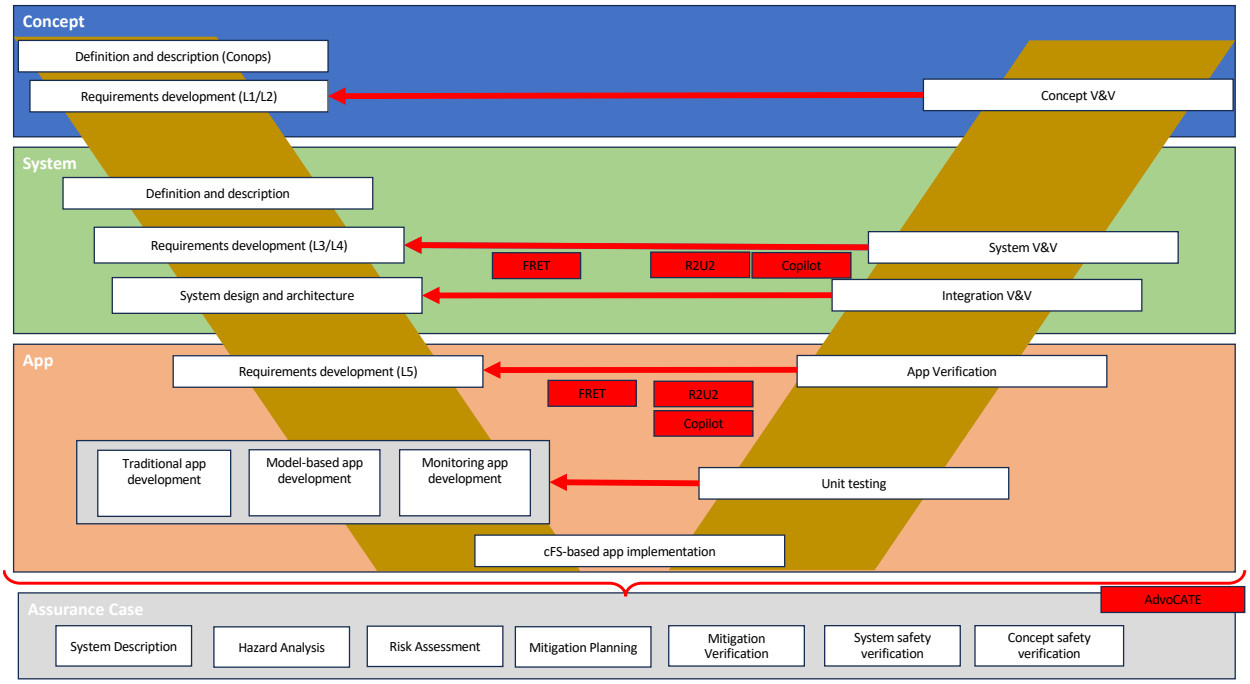

Fig. 1 Troupe development and verification life-cycle.

A. Project Management

While the Troupe mission is set up as incubator for experimenting with new methods for developing spaceflight software, it is done in the context of NASA's requirements and guidelines for all spaceflight projects. As a result, Troupe follows the NASA Procedural Requirement (NPR) 7120.8A for project management. A project plan for Troupe was developed which includes a description of the goals and objectives, deliverables, management approach, monitoring and required reviews, and expected schedule and cost. As an example, the Troupe goals and objectives are shown in the Fig. 2 below.

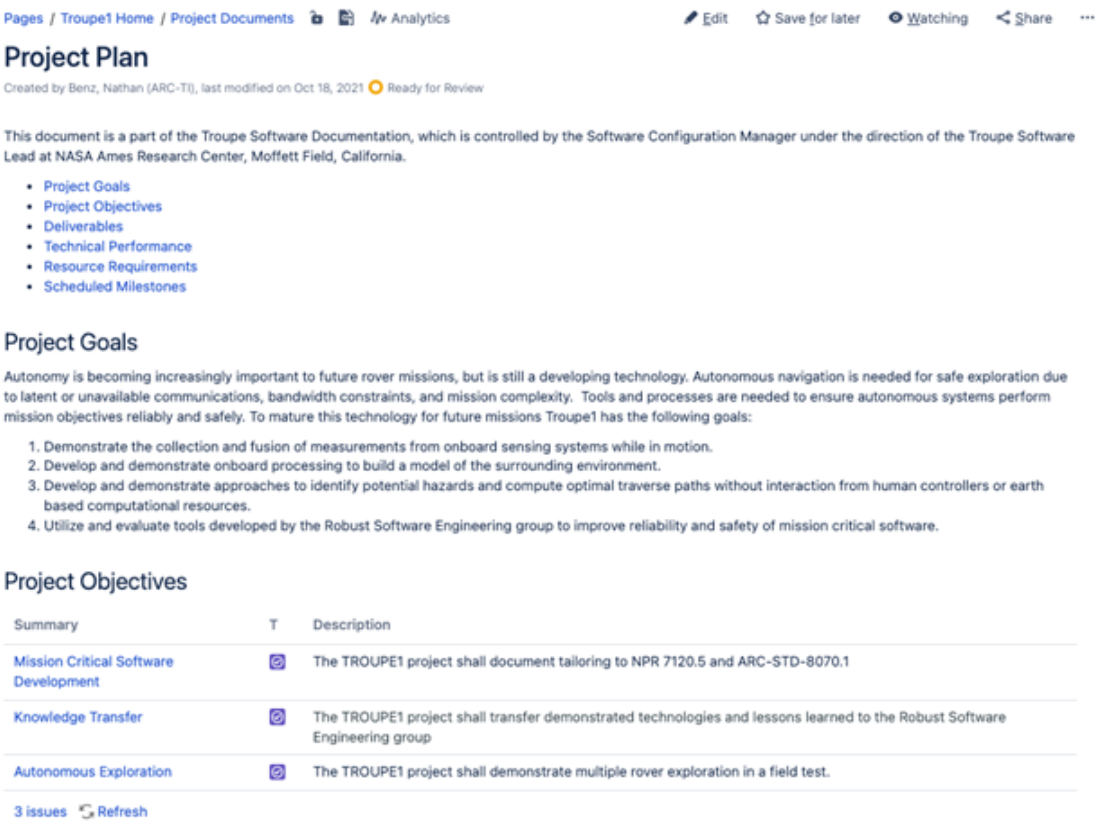

Fig. 2 An example of the Troupe project goals and objectives.

Typical NASA spaceflight mission utilize a "waterfall" management approach, while in industry typical software products are developed using agile approaches to project management. Troupe uses a hybrid approach. Milestone reviews such as Mission Concept Review (MCR) and Critical Design Review (CDR) are scheduled using a waterfall approach, the actual product development utilizes two-week sprints, retrospectives, scrum planning, continuous integration, and epic burn down charts to plan and measure progress towards the milestone reviews. The goal is to adopt current best practices for agile project management into the traditional NASA product life-cycle model.

B. Systems Engineering

The system architecture design for the multi-rover mapping reference mission utilizes Model-Based System Engineering (MBSE) methods to define the interfaces, operating modes, functional decomposition, and system allocation of requirements. MBSE allows for a more formal and precise description of the system as opposed to a document-based description. This also provides a use-case for formal model checkers for V&V of the system requirements. As an example, instead of a traditional Interface Control Document (ICD) used to specify the data interfaces between software modules, a SysML Internal Block Diagram (IBD) is used as shown in Fig. 11. By specifying the interfaces this way, a contract-based approach to development can be adopted. In this approach, the interfaces between system elements are defined and be queried and verified by the model. A component developer can implement a component according

to this machine readable specification, removing the ambiguity often encountered by a document-based specification written is unstructured natural language.

C. Software Development

Troupe is a software-focused project, and the reference mission was chosen to provide an example of distributed network of autonomous agents which provides a challenging use-case for the V&V tools to utilize. The development process follows NPR 7150.2D to satisfy NASA software engineering requirements for flight missions. This includes the development of software cost estimate, software architecture and design, configuration management plan, testing plan, risk management and requirements management. Software requirements and development tasks are captured using Atlassian's Jira software. Bi-directional traceability between requirements and functional code is established using Bitbucket. Continuous integration and testing are done using Atlassian Bamboo.

D. Verification and Validation

Troupe utilizes numerous tools to verify and validate software capabilities. These tools fall into two categories: Those developed by NASA Ames RSE and those acquired from outside of NASA. We used the following NASA Ames RSE tools within Troupe:

1) AdvoCATE to perform hazard analysis and build a safety case (dynamic runtime assurance);
2) FRET to elicit, formalize requirements and generate requirement-based test-cases;
3) CoCoSim with Kind2 to formally verify Simulink models;
4) Ogma/Copilot for runtime monitoring of C code;
5) R2U2 for runtime monitoring of C code.

Regardless of the app type, whether they are reused, custom developed or automatically generated, we perform unit tests for all apps. Where possible, we utilize FRET to formalize requirements and generate R2U2 or Ogma/Copilot monitors and in that way we can perform L5 requirement verification on the app level.

We perform automated integration tests and manual testing to verify the system design and architecture. We utilize FRET to formalize the L4 safety requirements and generate R2U2 or Ogma/Copilot monitors that are used to verify system-level properties, where possible.

For the model-based apps, we first perform functional decomposition based on the L5 requirements, detailed app design in Simulink, and then we formalize requirements using FRET and export Lustre specifications that can be given as input to the CoCoSim tool. CoCoSim can then be used for model checking the Simulink models with the Kind2 tool. In the case that model checking is proven infeasible, we perform test-case generation using FRET and CoCoSim.

Troupe also employs off-the-shelf (OTS) tools for verification and validation. Below are each of the OTS tools Troupe uses.

- AirSim [26] is an open source 3D simulator produced by Microsoft that allows Troupe to perform high fidelity subsystem testing. Powered by the versatile Unreal Engine, AirSim allows the Troupe team to create a variety of high-fidelity terrain, create event triggers to simulate events, and have multiple rovers in dynamic or scripted scenarios. We connect the Troupe rovers to AirSim to perform subsystem testing.
  Troupe is able to simulate rover movement through a custom port of the embedded hardware controller, ArduRover. Troupe has simplified much of ArduRover's capabilities to create a smaller app called ArduPort, which simply translates cFS messages to MavLink messages and sends the messages to the motor controller. AirSim can receive MavLink messages natively and simulate vehicle responses. This allows Troupe to generate simulator interactions that mirror real communication interactions.
  In addition to simulated movement, Troupe rovers also require localization data to estimate their current positions. In hardware, Troupe uses a Decawave DWM1001 radio to triangulate its position with anchor radios. In simulation, Troupe has developed a custom companion app to AirSim that allows AirSim to broadcast simulated position data. AirSim is also capable of providing camera images to simulate the rovers capturing images. The simulated camera interfaces with the Troupe software exactly as a hardware camera would. From a software perspective, the simulated and real cameras are identical. No code logic changes are needed.
- Atlassian Bamboo [27] is an automated testing pipeline that Troupe uses to execute unit tests and run regression tests on the code base. Bamboo is an essential tool for verification and validation as it can be configured to execute hundreds of tests automatically after each push, allowing Troupe to spot any instances of broken code before they are merged into the main branch.

III. Concepts of Operations

Troupe consists of a Mission Operations System (MOS) and a Rover Swarm (RS), which is comprised of 4-10 rovers. Rovers are tasked with mapping an area of interest. The coordination and collaboration scheme is based on the Dynamic Zonal Relay (DZR) and Sneaker-Net Relay (SNR) algorithm proposed in [28]. Rovers are assigned one of four roles: Lander, Tail, Relay, Lead. There is only one lander rover tasked with acting as the base station. This rover communicates with the ground station and is responsible of receiving telemetry and data from the RS and transmitting them to ground. At deployment, the DZR stage commences by assigning each rover a role and a designated zone. Rovers are tasked with traversing to their zones and mapping their areas. After a rover is done mapping its zone, it acts as a relay rover until the SNR phase commences. Once the initially assigned zones have been mapped and compiled by all rovers, the SNR phase begins to extend the operation area. The lead rover traverses to the head of the formation and characterizes new zones that are potentially beyond communication range with the lander. All follower relay rovers maintain a coordination and communication stream between their direct leader and direct follower. These rovers transmit telemetry and map data back to the lander once mapping is complete. They also transmit commands and confirmation messages forward towards the leader. The tail rover is in essence a relay rover with the lander as its direct follower.

Fig. 3 illustrates the highest level components of Troupe. Troupe is composed of two components: The Mission Operations System (MOS) and the Rover Swarm (RS). Here the rover swarm is displayed as one block; however, the rover swarm is made up of between 4 and 10 rovers, depending on Troupe resources.

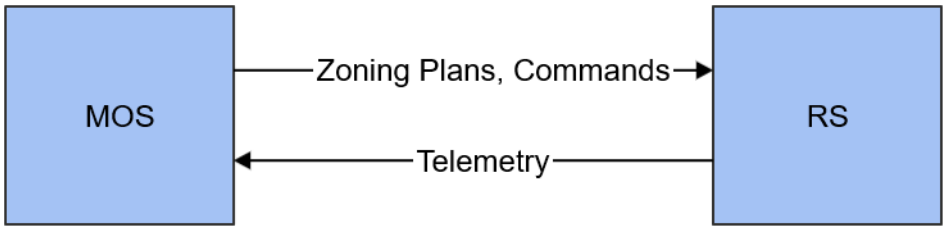

Fig. 3 High-level Troupe System Overview.

IV. Troupe System Hardware

A. Electrical Design

The Troupe rovers are designed to be small, lightweight, and simple to assemble in batches. A single rover can be powered by one 7.4V 2-cell Lithium-polymer battery. The accompanying power system reflects the Troupe rovers' size. The components are designed to sit on top of an off-the-shelf remote control (RC) truck chassis. The main electronic components include the battery, voltage regulator, Raspbery Pi 4B computer, a USB hub, an Intel RealSense D435 stereo camera, a Decawave DWM1001 ultra wide-band (UWB) radio module, a BeagleBone Blue computer, the main drive motor, and the steering servo. The rover power system is shown Fig. 4. The BeagleBone Blue will simply be referred to as the avionics computer.

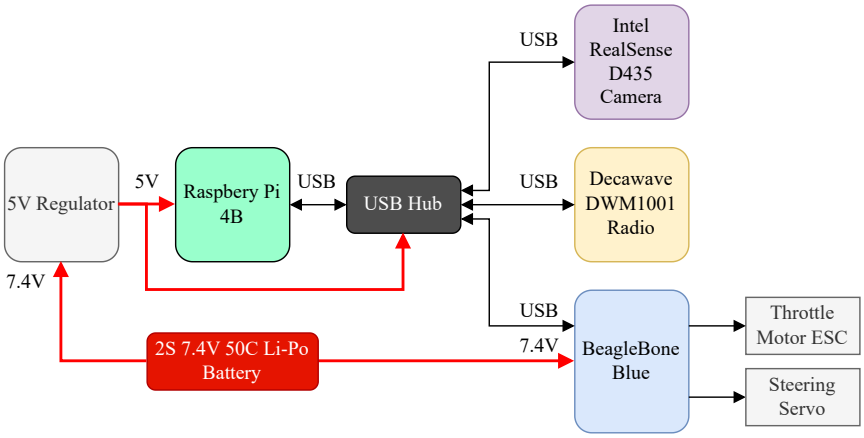

Fig. 4 The power system diagram for a typical Troupe rover.

By itself, a Raspberry Pi cannot supply enough power to all of the listed components. The combined current draw of the camera, avionics computer, and radio module is approximately 1.4A and exceeds the current output of the Raspberry Pi (1.2A). As such, an externally powered USB hub is used to connect the components and help supply enough current. USB connections were chosen so that a rover could be easily taken apart and put back together again for testing. This also allows individual components to be tested with auxiliary computers for the purposes of testing and debugging.

B. Mechanical Design

The mechanical design for the Troupe rover is based on a 3D-printed and laser-cut platform that connects to the off-the-shelf RC truck chassis. Most of the electronic components reside within the 3D-printed platform. An early prototype of the Troupe rover is shown in Fig. 5, with improved iterations to follow.

Fig. 5 A prototype Troupe rover with a 3D-printed component box connected to an RC truck chassis.

One of the features of the 3D-printed design includes puzzle-like edges for each piece. The edges fit together with a friction fit. This allows the rover to be easily taken apart and put back together. Each piece can to be modified individually as needed. The front-facing wall has a cut out for the stereo camera and the side walls have cut outs for air-flow. There are also openings in other panels for power and signal wires. The components like the Raspberry Pi are attached to the 3D-printed platform with standoffs and screws, providing stability during navigation.

C. Using Ultra-Wide-Band Radios for Real-Time Location

Each Troupe rover has a single ultra wide-band (UWB) radio module. The radio is used as part of a real-time location system (RTLS) and provides accurate position measurements of a rover's location. The software interfaces with the radio module (shown in Fig. 7), through the Decawave software API. UART communication functions from the API are called to read position measurements in x, y, and z in millimeters as measured by the radio module. Position measurements are accompanied by a quality factor that indicates the percent accuracy of the measurement. The position measurement is calculated onboard the radio module and is based on a two-way-ranging (TWR) location engine. The location engine can both calculate and *estimate* the radio module position.

Each radio module can be configured as an "anchor" or "tag." Anchors are stationary with respect to the local environment and are used by the tag to measure its position. Unlike the anchors, the tags can move. The radio modules onboard the rovers are configured as tags. At least three anchors must setup in an area in order to measure the tag, and by extension the rover's position. Fig. 6 shows a two dimensional example layout of four anchors and one tag.

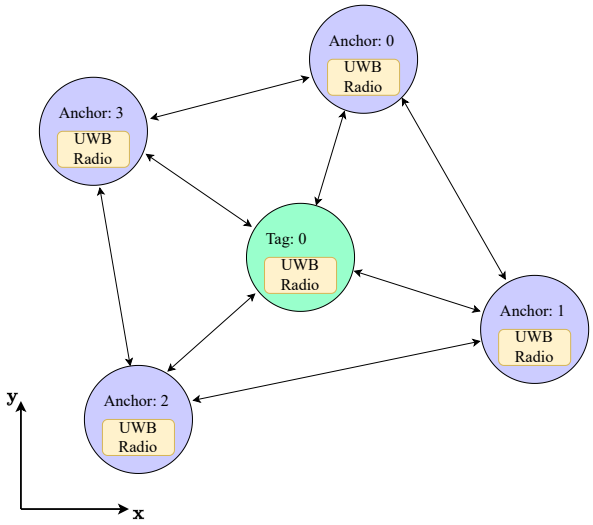

Fig. 6 An example configuration of UWB radios. Four radios are configured as anchors and one is configured as a tag.

V. Troupe System Software

A. Core Flight System

The Troupe project uses the NASA's core Flight System (cFS). cFS is a platform and project independent reusable software framework [29]. It consists of a set of reusable software applications. The three main components of the cFS architecture include: 1) a dynamic run-time environment, 2) layered software design, and 3) a component based design. cFS is comprised of a series of applications. Certain apps such as scheduling (SCH), the core-flight executive, command ingest (CI), house keeping (HK), and telemetry output (TO) are required. Other apps can be added as necessary, depending on the project requirements.

cFS is built and cross-compiled on a host computer and run on a Rasbperry Pi onboard the Troupe rover. An example diagram of the Troupe rover cFS apps is shown in Fig. 7. Some apps operate on internal basis only, while others can interface with the rover hardware. For example, the Arduport app interfaces with the avionics computer over USB. Messages can be sent to the Arduport app either to retrieve system status information or to command the rover to move.

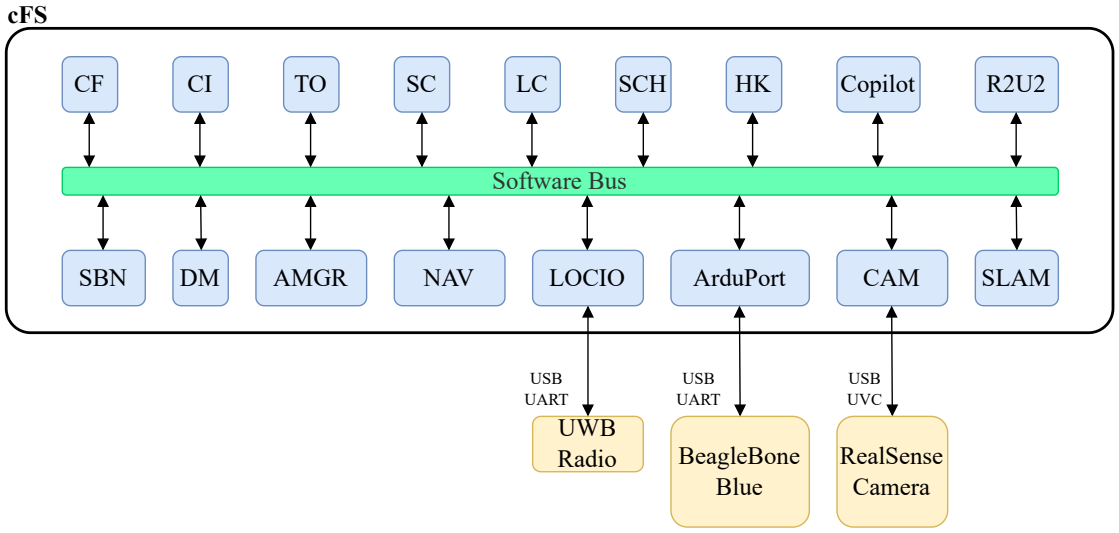

Fig. 7 The cFS applications comprising Troupe.

B. Applications Developed for Troupe

The list of applications developed specifically for Troupe include the following:

1. DM: Decision making

DM utilizes system measurements and events to determine the system phase of operation using a hierarchical finite state machine (H-FSM). DM is made up of three main layers: The event system layer, the H-FSM, and the parametrization layer. The event system layer receives flags that reflect system status and threshold checks from the AMGR app and contains logic that converts these flags into events that govern H-FSM transitions. The H-FSM layer is responsible of receiving system events as an input and then triggering state transitions as a result. The parametrization layer monitors the H-FSM state the system is in and generates a list of parameters used to conFig. other apps for that phase of operation.

2. AMGR: Agent Manager

AMGR serves two purposes. Its first purpose is to act as an interface between the DM app and other apps of the Troupe system. Namely, it collects the required measurements and performs the flag logic necessary to status the set of flags needed by DM. The second purpose of AMGR is to manage the rover high level role and interface with MOS.

3. NAV: Navigation

NAV generates path plans for the rover to follow based on the assigned zone within DZR+SNR and the occupancy map generated by SLAM. It also houses the controllers used for waypoint following and an extended kalman filter for localization.

4. LOCIO: Real-Time Location System via Radio

LOCIO interfaces with the Decawave DWM1001 radio module. It configures the radio module as a tag and provides the location of the rover with respect to the anchors. Like other apps, LOCIO can subscribe to different types of messages on the software bus (SB). A command message can be sent to LOCIO over the SB with an accompanying command code. For instance, one command code can specify that LOCIO executes a No-Op, another can request a position measurement, and another can request a list of all the anchors positions in the area.

5. Arduport: MAVLink Communication with ArduPilot

Arduport is in charge of sending and receiving MAVLink messages over USB UART to and from the avionics computer. The Raspberry Pi sends command messages for arming, disarming, steering, and throttle. The avionics computer runs a custom version of ArduPilot which allows it to respond to custom commands while periodically publishing status messages. Some useful status messages include system status, vehicle attitude, battery status, and other MAVLink messages.

6. CAM: The Camera Application

The CAM app interfaces with the Intel RealSense D435 stereo camera. This camera is used by SLAM and accepts commands over the software bus to capture images.

7. SLAM: Simultaneous Localization and Mapping

The SLAM app generates an occupancy map of the surrounding environment along with a list of hazards (obstacles), their locations, and time-stamps. It ingests range and angles measurements from the LIDAR cameras and also receives location data from LOCIO to assist in the localization process. Occupancy map construction is performed using the SLAM algorithm library provided by Mathworks. New images are ingested and compared to previously scanned ones. Based on how the images align and after some optimization, SLAM is able to construct a map of its surrounding environment and determine its location.

C. Applications following Model-Based Design Approach

During the software development process of the LADEE spacecraft, the NASA ARC Flight Software group developed the Simulink Interface Layer (SIL) tool that allow model-based Code Generation from Simulink models directly into cFS applications [30]. Model-based development utilizes a mathematical and visual approach for the development of software for complex systems and behaviors. This is coupled with code generation push button capability that supports two way traceability between models used in development and the generated code. Such a development approach offers multiple advantages [30–32] as it allows

- Early prototyping and refinement of software requirements.
- Design and algorithm development in a visual and easy-to-read environment.
- Rapid test deployment and simulation environment design.
- One button code generation with two way traceability between models and generated code.
- Direct integration with cFS heritage software using the SIL tool.

Multiple apps of the Troupe system contain complex behaviors and interactions that would benefit from such a development approach. Namely, DM, AMGR, NAV, and SLAM. Fig. 8 shows an example of generated code from the DM model and the model-code traceability. Fig. 9 illustrates the input and output ports of the SLAM model-based design. These ports are converted from Simulink buses to cFS buses on code generation using the SIL tool.

The model-based development process in Troupe follows the process outlined in Fig. 10. A conceptual design is first constructed from the system requirements. This includes the design and development of solutions that achieve the required functionality under the imposed requirements. The conceptual design is then fully realized in an implementable manner in the embodiment design stage. Model-based implementation is then performed in Simulink. This allows rapid prototyping, model-in-the-loop testing, and early model validation and assurance. Code is then generated from the implemented model and integrated into Troupe cFS. Traditional app testing and V&V follows from that point.

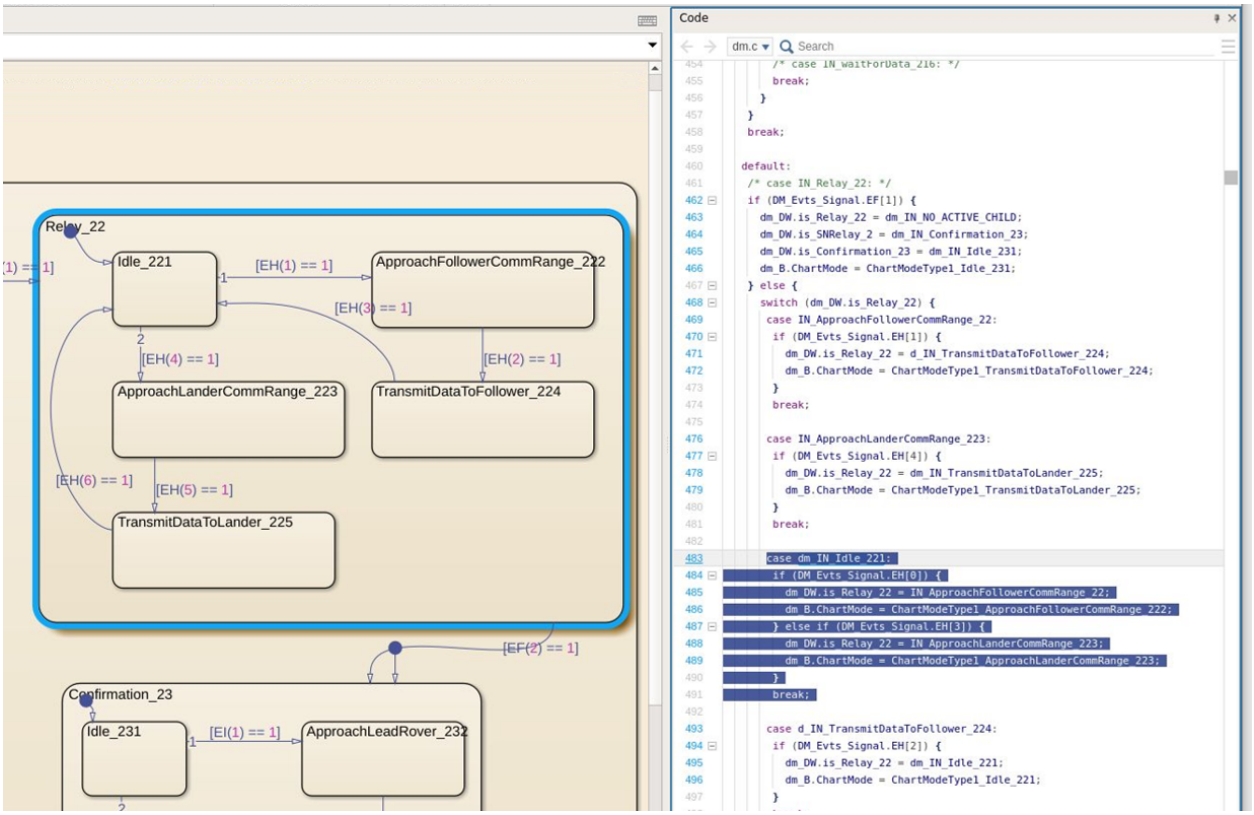

Fig. 8 Example of traceability between model-based DM implementation and generated code.

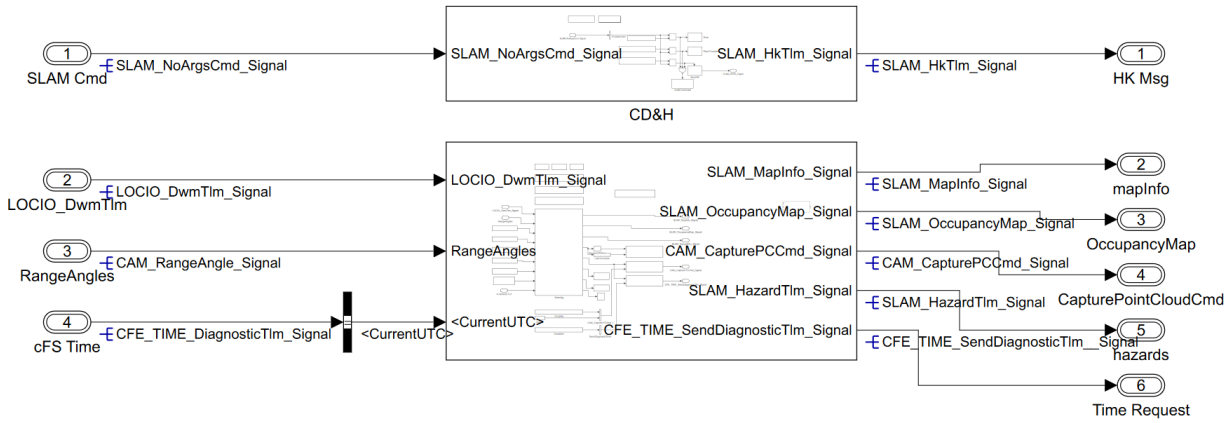

Fig. 9 The highest level of the SLAM Simulink model.

D. User-Level Application Layer Architecture SysML

One issue with analyzing cFS-based systems is the lack of a clear architecture of the system. All the connections between different apps go through the common software bus. There is a lot of packets being published, and very few are actually consumed by the user apps. Furthermore, each of those packets contains many attributes, and not all of those are being used, at least not by the same app. To really understand the interaction between components, we specified a detailed architectural design in SysML (Fig. 11). In this detailed design, we indicate which variables from each of the exchanged packets are consumed by the subscribing application. For example, SLAM is using x_pc, y_pc and imageID

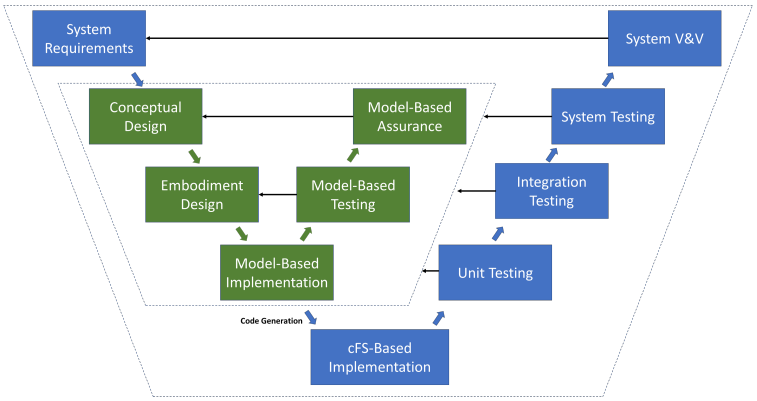

Fig. 10 Model-based app development process.

from the CAM_PointCloud payload, and seconds and sub-seconds from its header. This way we know for each app exactly which inputs it needs, and which outputs other apps need from it.

The information flow between user-level applications is important to understand and visualize. Seeing all the message subscriptions and underlying information flow between applications allows for simplifying applications' subscriptions and improving architecture efficiency. For example, the NAV app could be configured to publish filtered x and y positions, where as the LOCIO app publishes averaged and *raw* position values. An application such as a run-time monitor might require raw sensor values in addition to or instead of filtered values and therefore must subscribe to LOCIO.

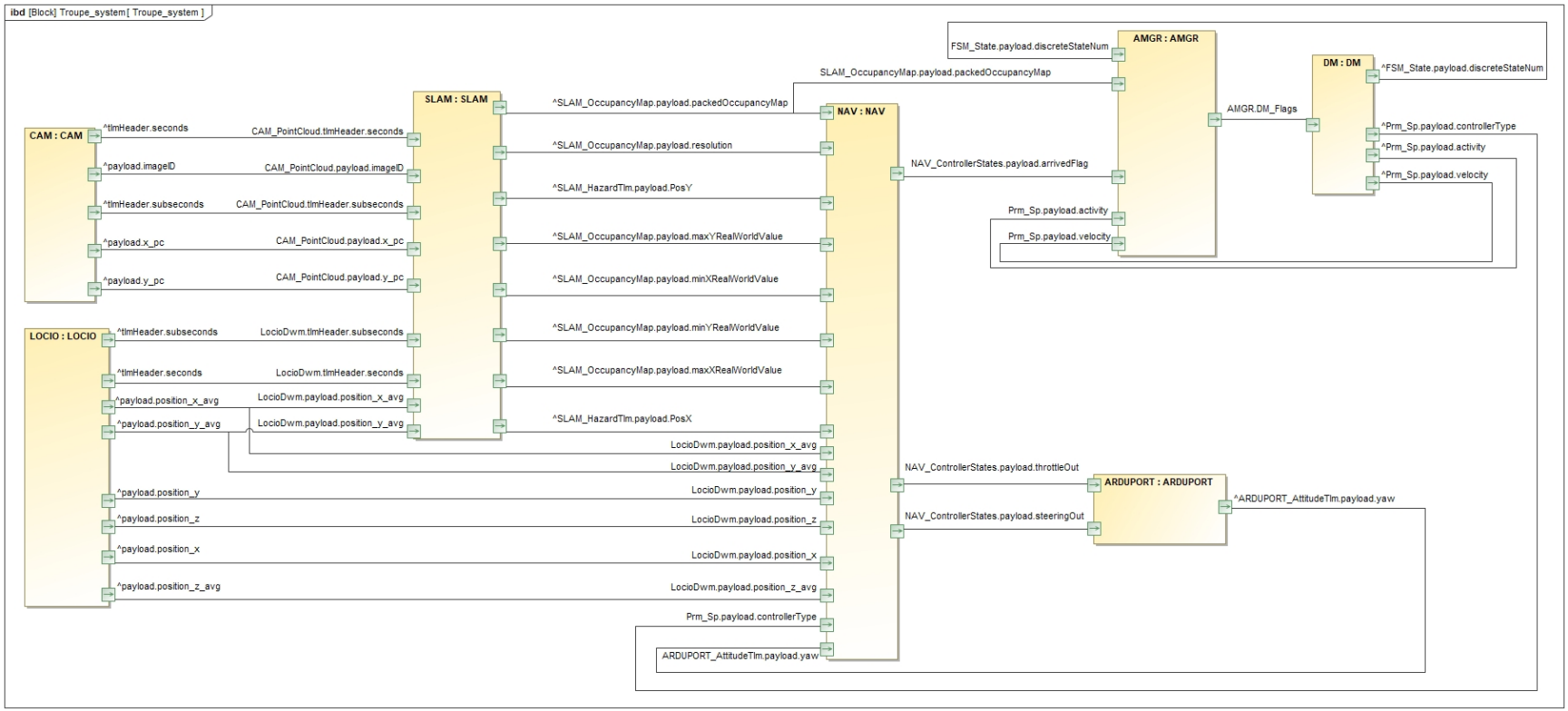

Fig. 11 Detailed architectural design of the Troupe user-level application layer.

VI. Lessons Learned and Future Work

The Troupe project development cycle revealed important insights that will help improve the rover swarm software, hardware, testing, verification, and validation. Below are a list of insights gained during development:

- **Model-Based Development and code generation:**
  - The rovers in Troupe are composed of both traditional and model-based apps for which the code is automatically generated. The first challenge we had in this hybrid setting is keeping in sync the model-based and traditional apps and their interfaces. Since we opted to use Simulink to model entire cFS apps and generate complete cFS apps using SIL and ECI cFS plugins, maintaining the interfaces between the apps meant a lot of manual work in connecting the traditional app headers and the Simulink inputs and busses.
  - Developing and testing in isolation the model-based apps is generally done on different computers than the target platform. This led to some of the model-based code generated apps like SLAM to cause stack overflows during run time on the target platform. Furthermore, the code generation for a cFS application using MATLAB could take upwards of 9 minutes. This slowed down development of MATLAB code generated apps and made integration testing much more difficult.
  - Although we had multiple cFS apps built in Simulink, we were only able to perform verification in isolation for those apps due to the way they had to be structured for cFS app code generation to work. This meant that all the integration testing between the model-based apps had to be done on the generated code and not in the model-based setting.
- **Agile development:**
  - Doing agile development meant that we started development before we had a more detailed design. This means that we should keep updating the design as the development is progressing. However, at one point the gap between the detailed design and the development grew too big, and it became difficult to bridge it.
  - Some requirements were written that did not sufficiently tie into lower-level implementations. For example, early requirements called for periodic position measurements from LOCIO app as input to the NAV app, but the position update rate, signal quality factor, and other lower-level settings to fulfil the requirement were not specified. However, in general the Level 1-3 requirements helped guide the development process of Troupe and helped distinguishing tasks that could be parallelized. Future Troupe development will focus on tying unit tests and subsystem tests directly to requirements to help speedup correcting and writing sufficiently detailed requirements per requirement level.
- **Getting the interfaces right early on:**
  - Maintaining up-to-date data types and structs used by developers was found to be very important for testing and development of V&V tools/methods. FRET and CoCoSim both benefited from early definition of the app interfaces that allowed them to plan ahead for the support of those data types that were not supported before. For example, both FRET and CoCoSim teams realised early the heavy usage of vectore signals in the Simulink models, which allowed them to support importing requirements for vector-element signals. Both tools extended their capabilities and allowed for extensive testing and evaluation of these tools as a result of the Troupe rover swarm acting as a set of test platforms.

As the Troupe project at NASA Ames continues to apply V&V methods to distributed robotic systems, detailed lessons learned and best practices will be published. This manuscript will serve as a reference introduction to project's goals and initial approach to system verification of mission critical software.

A. Troupe Companion Papers

This sections lists the companion papers that provide more details on various stages of the Troupe project development and verification life-cycle, shown in Fig. 1:

- *Design, Formalization, and Verification of Decision Making for Intelligent Systems* [33]
- *Dynamic Assurance of Autonomous Systems through Ground Control Software* [34]

References

[1] Giannakopoulou, D., Mavridou, A., Pressburger, T., Rhein, J., Schumann, J., and Shi, N., "Formal Requirements Elicitation with FRET," *REFSQ*, 2020.

[2] "FRET: Formal Requirements Elicitation Tool," , 2023. URL `https://github.com/NASA-SW-VnV/fret`.

[3] Giannakopoulou, D., Pressburger, T., Mavridou, A., and Schumann, J., "Automated formalization of structured natural language requirements," *Information and Software Technology*, Vol. 137, 2021, p. 106590. https://doi.org/10.1016/j.infsof.2021.106590.

[4] Jahier, E., Raymond, P., and Halbwachs, N., "The lustre v6 reference manual," *Verimag, Grenoble, Dec*, 2016.

[5] Mavridou, A., Bourbouh, H., Garoche, P. L., Giannakopoulou, D., Pessburger, T., and Schumann, J., "Bridging the gap between requirements and simulink model analysis," *Joint 26th International Conference on Requirements Engineering: Foundation for Software Quality Workshops, Doctoral Symposium, Live Studies Track, and Poster Track*, 2020.

[6] Perez, I., Mavridou, A., Pressburger, T., Goodloe, A., and Giannakopoulou, D., "Automated Translation of Natural Language Requirements to Runtime Monitors," *Tools and Algorithms for the Construction and Analysis of Systems*, edited by D. Fisman and G. Rosu, Springer International Publishing, Cham, 2022, pp. 387–395.

[7] Katis, A., Mavridou, A., Giannakopoulou, D., Pressburger, T., and Schumann, J., "Capture, Analyze, Diagnose: Realizability Checking Of Requirements in FRET," *Computer Aided Verification: 34th International Conference, CAV 2022, Haifa, Israel, August 7–10, 2022, Proceedings, Part II*, Springer, 2022, pp. 490–504.

[8] Mavridou, A., Katis, A., Giannakopoulou, D., Kooi, D., Pressburger, T., and Whalen, M. W., "From Partial to Global Assume-Guarantee Contracts: Compositional Realizability Analysis in FRET," *Formal Methods*, edited by M. Huisman, C. Păsăreanu, and N. Zhan, Springer International Publishing, Cham, 2021, pp. 503–523.

[9] "CoCoSim: Contract-based Compositional verification of Simulink models," , 2023. URL https://github.com/NASA-SW-VnV/cocosim.

[10] Bourbouh, H., Garoche, P.-L., Garion, C., Gurfinkel, A., Kahsai, T., and Thirioux, X., "Automated analysis of Stateflow models," *21st International conference on logic for programming, artificial intelligence and reasoning (LPAR 2017)*, Vol. 46, 2017, pp. 144–161.

[11] Bourbouh, H., Garoche, P.-L., Loquen, T., Noulard, É., and Pagetti, C., "CoCoSim, a code generation framework for control/command applications: An overview of CoCoSim for multi-periodic discrete Simulink models," *Embedded Real Time Systems (ERTS) 2020*, , No. ARC-E-DAA-TN74591, 2020.

[12] "IKOS: Inference Kernel for Open Static Analyzers," , 2023. URL https://github.com/NASA-SW-VnV/ikos.

[13] Brat, G., Navas, J. A., Shi, N., and Venet, A., "IKOS: A framework for static analysis based on abstract interpretation," *Software Engineering and Formal Methods: 12th International Conference, SEFM 2014, Grenoble, France, September 1-5, 2014. Proceedings 12*, Springer, 2014, pp. 271–277.

[14] Perez, I., Dedden, F., and Goodloe, A., "Copilot 3," Tech. Rep. NASA/TM–2020–220587, NASA Langley Research Center, April 2020.

[15] Pike, L., Goodloe, A., Morisset, R., and Niller, S., "Copilot: A Hard Real-Time Runtime Monitor," *Proceedings of the First International Conference on Runtime Verification (RV 2010)*, Lecture Notes in Computer Science, Vol. 6418, Springer, 2010, pp. 345–359.

[16] Pike, L., Wegmann, N., Niller, S., and Goodloe, A., "Copilot: monitoring embedded systems," *Innovations in Systems and Software Engineering*, Vol. 9, No. 4, 2013, pp. 235–255.

[17] Ogma, https://github.com/nasa/ogma, 2023. Accessed: 2023-06-01.

[18] Denney, E., and Pai, G., "Tool Support for Assurance Case Development," *Automated Software Engineering*, Vol. 25, No. 3, 2018, pp. 435–499.

[19] "GSN Community Standard Version 3," Tech. rep., Assurance Case Working Group of The Safety-Critical Systems Club, May 2021. URL https://scsc.uk/r141C:1.

[20] Authority, C. A., "Bowtie risk assessment models," , 2019.

[21] Rozier, K. Y., and Schumann, J., "R2U2: Tool Overview," *RV-CuBES 2017. An International Workshop on Competitions, Usability, Benchmarks, Evaluation, and Standardisation for Runtime Verification Tools, September 15, 2017, Seattle, WA, USA*, 2017, pp. 138–156. URL http://www.easychair.org/publications/paper/Vncw.

[22] Reinbacher, T., Rozier, K. Y., and Schumann, J., "Temporal-Logic Based Runtime Observer Pairs for System Health Management of Real-Time Systems," *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS*, LNCS, Vol. 8413, Springer, 2014, pp. 357–372.

[23] Geist, J., Rozier, K. Y., and Schumann, J., "Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems," *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*, 2014, pp. 215–230. https://doi.org/10.1007/978-3-319-11164-3_18, URL http://dx.doi.org/10.1007/978-3-319-11164-3_18.

[24] Schumann, J., Rozier, K. Y., Reinbacher, T., Mengshoel, O. J., Mbaya, T., and Ippolito, C., "Towards Real-time, On-board, Hardware-supported Sensor and Software Health Management for Unmanned Aerial Systems," *International Journal of Prognostics and Health Management*, 2015, p. to appear.

[25] Hejase, M., and Banerjee, P., "SafeMAP: Safe Multi-Agent Planning framework based on Dynamic Probabilistic Risk Assessment," *Annual Conference of the PHM Society*, Vol. 14, 2022.

[26] "AirSim: Welcome to AirSim," , 2023. URL https://microsoft.github.io/AirSim/.

[27] "Bamboo: Continuous Integration & Deployment - Atlassian," , 2023. URL https://www.atlassian.com/software/bamboo.

[28] Vaquero, T., Troesch, M., and Chien, S., "An approach for autonomous multi-rover collaboration for mars cave exploration: Preliminary results," *International Symposium on Artificial Intelligence, Robotics, and Automation in Space (i-SAIRAS 2018). Also appears at the ICAPS PlanRob*, 2018.

[29] NASA, "core Flight System," , 2021. URL https://cfs.gsfc.nasa.gov.

[30] Cannon, H. N., "flight software for the ladee mission," *Aerospace Guidance and Control Systems Committee Meeting# 116*, 2015.

[31] Ricco, A. J., Santa Maria, S. R., Hanel, R. P., and Bhattacharya, S., "BioSentinel: a 6U nanosatellite for deep-space biological science," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 35, No. 3, 2020, pp. 6–18.

[32] Pires, C., "Modular Infrastructure for Rapid Flight Software Development," Tech. rep., 2010.

[33] Hejase, M., Katis, A., and Mavridou, A., "Design, Formalization, and Verification of Decision Making for Intelligent Systems," AIAA/Scitech, 2024.

[34] Sljivo, I., Mavridou, A., Schumann, J., Perez, I., Vlastos, P., and Carter, C., "Dynamic Assurance of Autonomous Systems through Ground Control Software," AIAA/Scitech, 2024.