17th International Conference on Space Operations, Dubai, United Arab Emirates, 6 - 10 March 2023.

SpaceOps-2023, ID # 352

Experience and Lessons Learned from Open Sourcing NASA AMMOS's Mission Control Software

Joshua S. Choi*

* *Program Area Manager, NASA AMMOS Mission Control System, NASA Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Mail Stop: 301-270, Pasadena, California, 91109, United States of America*, josh.choi@jpl.nasa.gov

Abstract

The National Aeronautics and Space Administration (NASA) Science Mission Directorate (SMD) sponsors the Advanced Multi-Mission Operations System (AMMOS), which is a set of tools and services that provide reliable, reusable, and repeatable space operation capabilities to as many NASA missions as possible. AMMOS supports NASA's goal of achieving greater science at lower costs. Mission Control System (MCS) is one of the four functional elements under NASA AMMOS, and it includes mission-proven software such as the AMMOS Mission Data Processing and Control System (AMPCS). The Multimission Ground Systems and Services (MGSS) Program at the NASA Jet Propulsion Laboratory (JPL) manages the NASA AMMOS.

To broaden their reach and increase their adoption at other NASA centers and partner organizations, MGSS is releasing the following set of NASA AMMOS mission control software as open source, as early as June 2023: AMPCS, Mission Control Web Service (MCWS), Open Mission Control Technologies (Open MCT) for MCWS, Spacecraft Language Interpreter and Collector II (SLINC II), Command Translation Subsystem (CTS), Telecommand Utilities (TCU), and Standards Support System (SSS).

There are several challenges in open sourcing this software set. The code, its documentation, and accompanying data contain controlled unclassified information that cannot be made public. Open sourcing must also comply with United States export control regulations. The software set has dependencies on other proprietary software that are not publicly available. The budget to perform the open sourcing work is small. None of the software set's maintainers have any previous experience hosting and running an open-source project or managing communities for such projects.

A team of 6 software engineers (1.4 full-time equivalents, or FTE) at JPL and NASA Ames Research Center (ARC) has been employing techniques to remove or replace the software set's non-releasable code, documentation, data, and private external dependencies. They are also refactoring the source structure and documentation to follow open-source software style conventions.

Open sourcing NASA's premier mission control software set will encourage more centers and partners to use it for their future space missions. They will gain the ability to extend the software themselves to satisfy their unique mission needs, inspect problems directly when they encounter them, audit its cybersecurity, repair defects quickly, and benefit from the greater open-source community support to use the product better. NASA AMMOS is also expected to benefit from this future community, as the latter contributes code and improvements to the software.

Keywords: Mission Control, Open Source, AMMOS, AMPCS, Software Engineering, Ground System

Acronyms/Abbreviations

AMMOS Instrument Toolkit (AIT); Advanced Multi-Mission Operations System (AMMOS); AMMOS Mission Data Processing and Control System (AMPCS); application programming interface (API); Ames Research Center (ARC); Assembly, Test, and Launch Operations (ATLO); Common Access Manager (CAM); Change Control Board (CCB); Consultative Committee for Space Data Systems (CCSDS); CCSDS File Delivery Protocol (CFDP); Communications Link Transmission Unit (CLTU); Cybersecurity & Privacy Division (CSPD); Command Translation Subsystem (CTS); controlled unclassified information (CUI); Deep Space Network (DSN); Engineering, Health and Accountability (EHA); Event Record (EVR); flight-ground interface control document (FGICD); Free and Open Source Software (FOSS); full-time equivalents (FTE); fiscal year (FY); Ground Support Equipment (GSE); Goddard Space Flight Center (GSFC); integrated development environment (IDE); Instrument Data System (IDS); Jet Propulsion Laboratory (JPL); Mission Change Request (MCR); Mission Control System (MCS); Mission Control Web Service (MCWS); Mission Design and Navigation (MDN); Multimission Ground Systems and Services (MGSS); Mission Planning, Sequencing and Analysis (MPSA); National Aeronautics and Space Administration (NASA); Near Earth Network (NEN); National Institute of Standards and Technology (NIST); Open Mission Control Technologies (Open MCT); representational state transfer (REST); sensitive but unclassified (SBU); Standard Formatted Data Unit (SFDU); Special Function Gateway (SFG); Space Link Extension (SLE); Spacecraft Language Interpreter and Collector II (SLINC II); Science Mission Directorate (SMD); Space Network (SN); Standards Support System (SSS);

Telecommand Utilities (TCU); Test Like You Fly, Fly Like You Test (TLYF/FLYT); Telemetry, Tracking, and Command (TTC); user interface (UI); verification and validation (V&V).

1. Introduction

National Aeronautics and Space Administration's (NASA) Advanced Multi-Mission Operations System (AMMOS) provides software systems and services under four disciplines [1]: Mission Design and Navigation (MDN), Mission Planning, Sequencing and Analysis (MPSA), Mission Control System (MCS), and Instrument Data System (IDS). The MCS element of NASA AMMOS provides ground segment software that fills the following space mission needs:

1. Connect to and transfer data to and from ground stations, such as that of Deep Space Network (DSN), using Consultative Committee for Space Data Systems (CCSDS) Space Link Extension (SLE) and Near Earth Network (NEN)/Space Network (SN) protocols;
2. Process spacecraft telemetry and station monitor data in both real-time (i.e., live) and recorded (i.e., replay) mode. Supported data types include:
    a. CCSDS transfer frames,
    b. CCSDS space packets,
    c. CCSDS File Delivery Protocol (CFDP) data units,
    d. Engineering, Health and Accountability (EHA) channel data,
    e. Event Records (EVRs),
    f. custom data products, and
    g. DSN Special Function Gateway (SFG) monitor data;
3. Build, translate, and encode spacecraft telecommands (into CCSDS Communications Link Transmission Units, or CLTUs), and transmit them to ground stations for radiation;
4. Monitor, visualize, receive alerts (using preconfigured alarm settings), produce reports and analyze spacecraft state using onboard telemetry and ground station monitor data, in both real-time and on-demand;
5. Reliably manage—including archiving and querying—all telemetry, telecommand, ground station monitor, and ancillary data; allow flexible user queries based on extensive sets of parameters; and
6. Automation of routine spacecraft monitor and control activities.

NASA AMMOS MCS software is typically used across all phases of a space mission, from flight software development, through testbed environments and Assembly, Test, and Launch Operations (ATLO), and during mission operations. Its users consider this as one of the biggest advantages of using NASA AMMOS MCS, as the software supports the Test Like You Fly, Fly Like You Test (TLYF/FLYT) approach to space missions.

Licenses to use closed-source NASA AMMOS software can be obtained royalty-free by U.S. Government Agencies, third parties for research use, and—in some cases—partner U.S. Government Contractors.

This paper discusses why there is an interest in making available the core set of NASA AMMOS MCS software as open source, the challenges in open sourcing this software set, the goals of the open sourcing effort, the approach being taken by the open sourcing team, current progress, and lastly, the summary of the experience including key lessons learned thus far.

2. Why Open Source?

As its name indicates, the NASA Advanced Multi-Mission Operations System (AMMOS) is made available to multiple space missions so that each mission can develop its unique operational capabilities on top of the core, generic set that is provided out of the box. It is generally more cost-effective and lower risk for both individual missions and for NASA to tailor the elements of a proven, existing system rather than each mission developing its own solution from the ground up. AMMOS is a mature, mission-proven system, and NASA encourages its use by as many missions as possible.

If the source code of a software is freely available for possible modification and redistribution by its users, i.e., the software is open source, then there is greater potential value in the software. Some of the key benefits for its users are the complete freedom and control to extend the software and repair its defects on their own timeline, the ability to perform static code analysis, and access to crowdsourced support from the development community formed around the software project.

Every NASA space mission has its own unique requirements on the ground system. With planetary science missions in particular, meeting those requirements oftentimes involves extending the capabilities set of the existing,

reusable ground system, such as NASA AMMOS. NASA AMMOS is architected and designed to enable such extensions while generally avoiding modifications to its multimission core. However, there have been cases when missions needed such modifications in order to achieve the capabilities they desired.

When these modifications are needed in closed-source software, the only option for the mission is to submit a request to the software's development organization. Unless that organization is under the control or significant influence of the requesting mission, it is impossible to hard for the mission to control whether or not the capability will be delivered at all and—even if the capability is promised to be delivered—the timeline of the capability's availability. In the case of NASA AMMOS, the standard process is for such mission to submit a formal Mission Change Request (MCR), which is then reviewed by the Multimission Ground Systems and Services (MGSS) Program Office's Change Control Board (CCB). The CCB then approves or rejects the request, using as a key criterium whether the change benefits other missions also using NASA AMMOS or it is a unique capability particular to that mission (or just a small class of missions) without any benefit to the rest.

With open-source software, however, the mission has another open pathway for achieving the capability extensions that it needs. The mission can *fork* the software, i.e., create a copy of the source code for its own independent development, and add the capabilities directly. This obviously requires that the mission foots the bill for such development effort, but it is in full control of both the software changes and the timeline for implementing those changes. The mission avoids going through the process of submitting change requests, having to justify their validity, criticality, and urgency, then hoping for and waiting for their approval by the CCB, while being wholly dependent upon the development timeline of another organization.

The situation is similar when the mission encounters bugs in the software that need to be fixed. With closed-source software, a bug or defect is reported to the development organization. Other than trying to convince the other organization how critical and urgent the fix for it is and possibly pressuring them to put out a fix soon, there is little that the mission can do to ensure that the defect is repaired by a certain date. With open-source software, however, the mission team can allocate its own engineering resources if needed to fix the bug themselves, thereby reducing the risk of its mission success.

Another reason that open-source software has greater potential benefit for its users is that the users can directly audit the source code and analyze its vulnerability, such as that of cybersecurity. Ensuring that bad actors cannot jeopardize their missions is a prime concern for any space organization. NASA, for instance, has in place the Cybersecurity & Privacy Division (CSPD) under the Office of the Chief Information Officer which manages an Agency-wide program to correct known vulnerabilities and enforce other cybersecurity protocols. Other cybersecurity programs within the U.S. Government, such as the National Institute of Standards and Technology's (NIST) National Checklist Program, also inform the type of cybersecurity measures that need to be taken with software being used by U.S. space missions. When a mission needs to verify that its software does not have a known vulnerability, with open-source software, the mission can directly perform that verification using static code analysis, perhaps quicker than the time it may take to receive that confirmation from the development organization. There are many automatic static code analysis tool options available, including free ones, to use for this purpose; but manual auditing of code is also an option [2].

In many instances, online communities made up of both regular users and developers form around popular open-source projects, and these can be valuable resources. Via a project's issue tracker, message boards (online forums), mailing lists, and non-project-specific websites such as the popular Stack Overflow, modern-day software users are now accustomed to finding help from the user community instead of relying solely on the software's development organization. With open-source software, there is a greater opportunity of finding expert-level help in the project's online community thanks to the participation of enthusiastic software developers as well as other regular users.

NASA AMMOS already has a number of its individual software products available as open-source (https://github.com/NASA-AMMOS/). For example, the AMMOS Instrument Toolkit (AIT) is a fully open-source, Python programming language-based software that enables missions to handle downlink telemetry, sequencing, and commanding of a spacecraft's onboard instrument and CubeSats. As of February 2023, AIT's core project has 22 forks and 38 stars on GitHub (stars indicate the number of GitHub users that have expressed appreciation for the project and its maintainers). Although these numbers are small compared to some of the other popular open-source projects, considering that spacecraft development and flying them is a very unique niche among software developers, they can be considered significant. The number of forks possibly indicates that there are more than a handful of space missions doing their own independent development on the software to extend its capabilities and/or to fix bugs on their own. Other NASA (but not AMMOS) open-source software that have already developed strong online communities around their projects and have seen significant adoption outside of their own organizations are NASA Ames Research Center's (ARC) Open Mission Control Technologies (Open MCT, https://nasa.github.io/openmct/) and NASA Jet Propulsion Laboratory's (JPL) F' (or F Prime, https://nasa.github.io/fprime/).

The development organization also stands to gain from open sourcing their software. One possible major benefit is engineering economics [3]. For example, as is the case with many software projects, there is limited funding for software development in NASA AMMOS in comparison to the large number of improvements and bug fixes that exist in its product backlogs. By open sourcing, developers outside the organization are able to contribute, mainly through contributing code that are first evaluated by the project maintainers and, if deemed beneficial to the project and meets its quality standards, merged in. These external code contributions are almost always volunteer work that do not require financial compensation.

Overall, there are definite expected rewards for open sourcing NASA AMMOS software products. All of the benefits for the mission users described above can promote increased adoption of NASA AMMOS and its open-source products at more NASA centers, partners, and their missions. When more missions take advantage of NASA AMMOS, there will be less need to reinvent the proverbial ground system wheel, and that results in general cost-savings for the Agency. This is the ultimate driver for considering and taking on the challenge of open sourcing as many NASA AMMOS software as possible.

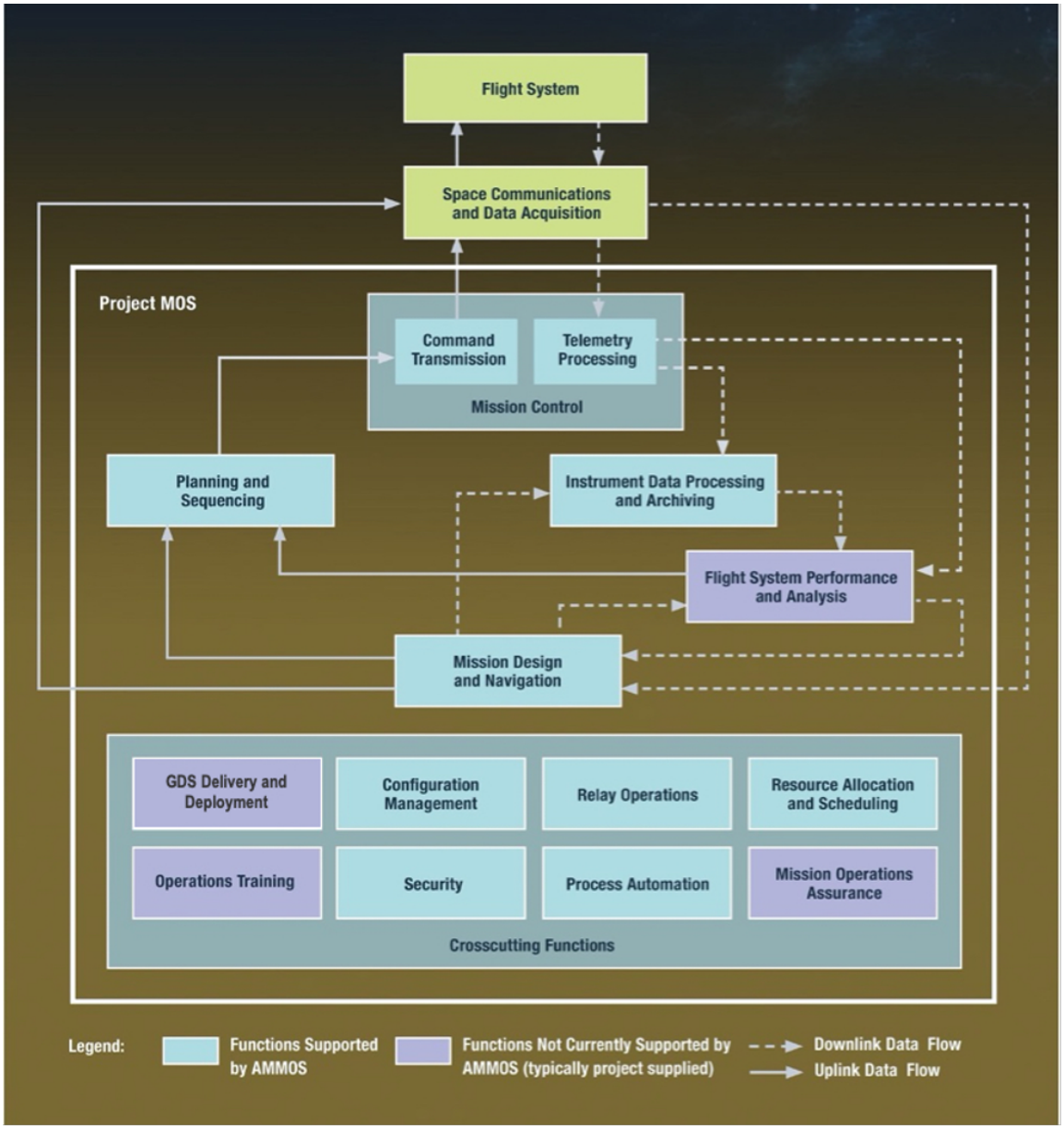

Fig. 1. High-level functional view of a flight project mission operations system and the four functional disciplines supported by NASA AMMOS.

3. Challenges

MGSS commissioned a task to fully open source a number of software products within the Mission Control discipline of NASA AMMOS during the fiscal year 2023 (FY23). (The four functional disciplines under NASA AMMOS, namely Mission Design and Navigation, Planning and Sequencing, Mission Control, and Instrument Data Processing and Archiving, map one-to-one to the four AMMOS elements previously mentioned [1]. See Fig. 1.)

A total of seven Mission Control software are being open sourced as part of this task:

1. AMMOS Mission Data Processing and Control System (AMPCS),
2. Mission Control Web Service (MCWS),
3. Open Mission Control Technologies (Open MCT) for MCWS,
4. Spacecraft Language Interpreter and Collector II (SLINC II),
5. Command Translation Subsystem (CTS),
6. Telecommand Utilities (TCU), and
7. Standards Support System (SSS)

AMPCS is the flagship AMMOS mission control software that handles telemetry processing, information monitoring, storage, query, CCSDS SLE protocols, and Ground Support Equipment (GSE) interfaces; it includes a Python-based automation toolkit; it provides commanding support including the user-interactive composition of commands, arguments, and fault injection. AMPCS supports all phases of a mission, enabling true TLYF/FLYT practices.

MCWS is the web service layer for AMPCS, its databases, and its message bus, providing access to mission engineering data entirely over a representational state transfer-based (i.e., RESTful) web application programming interface (API). It has the flexibility to allow more than just AMPCS to be attached as its information source. It also integrates with the NASA AMMOS Common Access Manager (CAM) to enforce authentication and authorization policies on its APIs.

Open MCT for MCWS is a plug-in software for integrating the generic Open MCT with AMPCS via MCWS. Open MCT itself is a graphical, web user interface (UI) that allows users to compose custom displays for analyzing telemetry information. Its flexible visualization capability is state-of-the-art in the space mission domain.

SLINC II, CTS, and **TCU** collectively provide the spacecraft telecommand validation, translation, encoding/decoding, and wrapping/unwrapping functions. Compared to other software and tools that perform the same or similar functions, this particular set of software is considered the gold standard for telecommand capability at JPL.

Fig. 2. Open MCT showing telemetry data from the NASA Mars Cube One (MarCO) spacecrafts, processed by AMPCS.

SSS is a collection of software libraries and applications that are used to handle, generate, and inspect Standard Formatted Data Unit (SFDU) files [4]. The aforementioned CTS and TCU have dependencies on this software.

Fig. 2 shows a sample view of Open MCT integrated with AMPCS and MCWS, displaying telemetry data from the NASA Mars Cube One (MarCO) CubeSats (MarCO A and B).

There are three major challenge areas in open sourcing the above set of software: (1) compliance with U.S. export control laws and regulations, in addition to identifying and handling controlled unclassified information (CUI) [5] and sensitive but unclassified (SBU) information, (2) ensuring that the newly open sourced software can be built, integrated, and run independently by the users in the public without the proprietary system environment and other software available only within JPL/NASA, and (3) producing adequate and helpful user instructions to accompany the software, with automatic tests that can be run by the users to quickly check the functional correctness of the installed and integrated system.

All 7 software projects above started out as closed source, and the original assumption was that they would remain so. Hence, although these software were purposed to be multimission and therefore agnostic toward particular missions (or classes of missions), some mission-specific information became embedded in the code and the code comments, as features were built in response to change requests from those missions. Especially in automated tests, such as unit tests, actual mission data and dictionaries (tables of telemetry and command format, structure and semantic definitions) became extensively used. In some parts of the code, snippets of a mission's flight-ground interface control document (FGICD) can be found in the code comments, as the developer tried to describe the rationale behind a certain algorithm which he implemented. Although we deemed that the mission control technology in these software themselves are safe from export control laws and regulations, some of the mission-specific information and data are restricted. Additionally, in some parts of the software, personally identifiable information can be found. Some secrets may have been hardcoded into code or the default configuration files. The entire codebase therefore must be scanned for and dealt with all of this unreleasable information and data, i.e., they need to be completely removed or at least replaced with public- and export-safe alternatives.

When there was no expectation that users and organizations outside JPL, specifically, other than the NASA AMMOS team at JPL, may be building the software from source on their own and in their own environments, there was not much concern with regard to which of the software dependencies need to be made freely available outside JPL. However, when MGSS first expressed the intention to open source AMPCS along with Open MCT for MCWS (which would give users the powerful visualization technology to use with AMPCS), we needed to determine which of their existing dependencies may be showstoppers toward that goal. We identified a number of dependencies that public users would need access to in order to build, integrate and run AMPCS independently. Those dependencies that are configuration-managed as standalone NASA AMMOS items are shown in Fig. 3. Not included in the figure are dependencies that are non-AMMOS proprietary libraries and whose code and binary packages are not yet available in the public domain; these include JavaCFDP [6] and SLE User. JavaCFDP is a software developed at the NASA Goddard Space Flight Center (GSFC) that provides CFDP functions. SLE User is commercial software, developed and sold by LSE Space GmbH, a Swedish Space Corporation company; it provides CCSDS SLE functions. AMPCS uses both under its hood to provide users with CFDP and SLE capabilities in the software.

Another major challenge is making sure that adequate instructions are provided either as part of the source code or in a separate, public documentation to assist the new user base, for them to build, integrate, and run all of the software. MGSS has requirements in place for the software projects that it sponsors to always provide Product Guides and User Guides to accompany each software release. A Product Guide is a technical document intended to give assistance to system administrators and software engineers administering and adapting a particular product. A User Guide, on the other hand, is a technical document intended to give assistance to people using a particular product in an operational setting. AMPCS and the other software being open sourced already have extensive sets of such documentation, but because of their volume, verbosity, and technical details (worthy of deep dives), we feel that if these are what the new users of our software have to rely on to get started, they will quickly become overwhelmed. Therefore, the open sourcing team is taking a fresh look at our software from the perspective of a completely new user from the public, so that they can produce a friendly, clear, and concise instruction set that will help those new users get started with the software easily and quickly. Preferably, this new set of documentation should follow the common conventions established by today's popular open-source projects (e.g. all user bootstrapping information contained in or branching out from the README.md file). Furthermore, many open-source projects provide automated tests that a user can run prior to operating the software to quickly smoke-test an integrated system. Creating and including such test(s) with our open-source software will further assist our new user base, but this also is additional work to be undertaken by the team and will prove especially challenging without the mission data and dictionaries that have previously been used for such purposes internally at JPL.

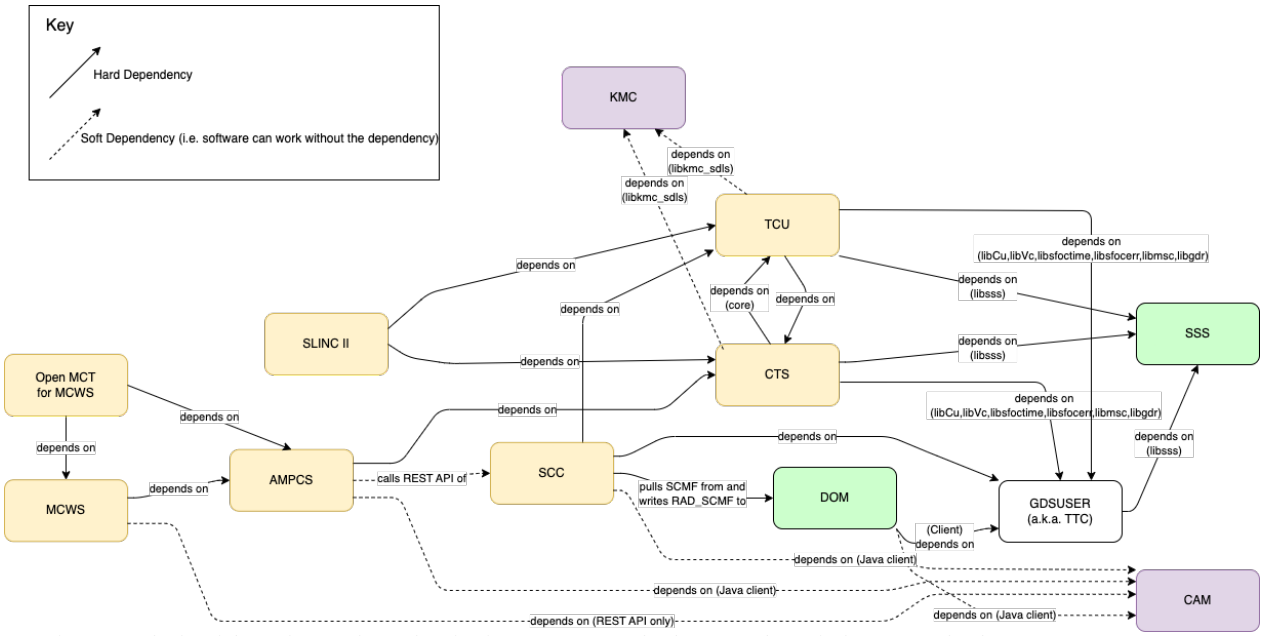

Fig. 3. Analysis of the software dependencies in MCS. Box color key: peach - Mission Control subsystem components, green - Data Management subsystem components, other - external to MCS

4. Goals

The results we are targeting to achieve by releasing AMPCS along with its supporting software of MCWS, Open MCT for MCWS, SLINC II, CTS, TCU and SSS as Free and Open Source Software (FOSS) are:

1. Wider adoption of NASA AMMOS's mission control solution, particularly the AMPCS software, at other NASA centers and partners;
2. Overall mission cost reduction for NASA, both as a result of 1. above and users having direct access to the source code for do-it-yourself (DIY) repairs and modification;
3. Missions and users have their needs met quicker, due to being less obstructed by NASA AMMOS's development and release timelines, resource limitations, processes, support availability, and other factors;
4. Fostering of a development community around the software, providing quality, crowdsourced resource that can possibly be tapped for future development and problem-solving on a volunteer basis, as well as generally improving the code on issues such as cybersecurity; and
5. Gradual reduction of resources being directly funded by NASA AMMOS in providing technical support and multimission development in response to mission and user requests, freeing up more money for new innovation.

5. Approach

The entire open sourcing effort was split into two phases. The main body of work was slated for FY23 (October 2022 through September 2023), but during its planning, we identified a major technical risk: users in the public will need access to all of the required software dependencies, as explained in the Challenges section above. Without these dependencies, users will not be able to build and run our mission control software. To retire this risk early, a task was performed in FY22 to identify all of these software dependencies and formulate a plan on how we can help the users get access to them. Many of the dependencies that we identified were already FOSS, and therefore no action was required on these items. Some were not, however, and therefore we needed to find a solution for them.

Two of those non-FOSS dependencies are NASA software. One is the set of telecommand utility libraries developed and maintained by the DSN and is part of the Telemetry, Tracking, and Command (TTC) System. Due to decisions made in the past, the CTS software has direct dependencies on these libraries, as shown in Fig. 3. The other NASA software dependency is GSFC's JavaCFDP. Neither of these software are open source.

A third non-FOSS dependency of concern is one that is commercially available. This is the previously mentioned SLE User software. AMPCS users that want to use its SLE capability will need to obtain this library and the required use license on their own.

After identifying these dependencies of concern, the team took action right away to kickstart the process of open sourcing the JavaCFDP code in collaboration with its development organization (GSFC) and to make publicly available the TTC libraries in their binary form for the x86-64 GNU/Linux architecture, also in collaboration with its development organization. The team plans to provide both of these dependencies to the users along with the MCS open-source software set.

When FY23 began, the MCS open sourcing team started a systematic scan (using text search tools and eye inspection) on the entire code base to catch any export control sensitive and CUI/SBU information and data. For secrets (i.e., passwords and other credentials) scanning, a number of tools were considered with the hopes of automating the job as much as possible. Git-secrets and Credential Digger were among those tools evaluated. However, the team concluded that it was more effective for the purpose of this task to rely on manual inspection rather than invest time and effort in setting up, configuring, and relying on these tools. These tools were too limited in their ability to detect sensitive information that are heavily context-based and domain-specific, such as those that exist in our mission control software.

During this process, when a piece of code or accompanying data is determined to be unreleasable, the team would then move that portion into a separate code repository. In effect, the previously single source code repository for each of the 7 software products now needed to be divided into two: one targeted to be moved to and hosted on GitHub (i.e., the open-source repository, or just repo) and another planned to be kept internally at JPL (i.e., private repo), on JPL's GitHub Enterprise instance. Any unreleasable code, data, and other types of information that existed in the previous repository would be moved over to the private repo. This procedure occasionally broke the code and made the software unbuildable. Regular refactoring of code was therefore needed, in addition to implementing any replacement code or adding generic data to repair the software to its working condition in the open-source repo.

The next step is to create one or more automated smoke tests that will accompany the software and that the user will run after building them to verify that everything is in working order. These tests should exercise the main functional paths in the software, namely, telemetry processing, command generation and transmission, real-time delivery of data, queries of archived data, and more.

The team will then produce user-friendly documentation that walks the user through the basic steps of how to build each software, install it, integrate it with the other software, configure it, run, and administer it. This user documentation should follow the current state of the practice for such information among the popular, modern open-source projects. This may mean that the team should opt to use the popular Markdown (.md) format or host the material on websites such as Read the Docs (readthedocs.org), which is where the AIT documentation is hosted.

AMPCS and the rest of the software being open sourced have already been part of the AMMOS System test and delivery process. This process is centered around AMMOS System's 3-releases-per-year cycle. All of the MCS software development tasks and product deliveries are aligned with the AMMOS System timeline and process. The fact that AMPCS and some of the other MCS software are now becoming open source does not necessitate deviating from or detaching our products from this existing arrangement, at least in the foreseeable future. Therefore, the first open-source versions of AMPCS, MCWS, and the rest of the software will still be released as part of an AMMOS System release, and that target release version is A32.1. This will allow the newly open sourced mission control software to be fully integration-tested as part of the regular AMMOS System. This is an additional quality assurance safety net because the verification of the AMMOS System is performed within environments that replicate those of current, actual JPL missions as closely as possible (i.e., very high-fidelity).

The team understands that getting to the point where the software's code is finally switched on to be public on GitHub is only the beginning of a significant, long-term effort. Here are some of the key responsibilities that the MCS organization must now begin to assume in order to maintain excellent open-source projects as well as to grow active communities around them:

- Publish product roadmaps that conform to AMMOS's multimission goals, make them public, keep them updated, and stick to them;
- Strictly enforce our multimission policy on all development contributions;
- Continuously update the software to keep up with the latest operating environments;
- Make security, quality, verification and validation (V&V) the projects' strong points;
- The team must continue to keep its reputation as having the foremost experts in the software; and
- Continuously maintain the gold-source repos.

In addition, over time the software projects should include all of the information that the open-source community will want to see, such as:

- Installation instructions,
- Quick start instructions,
- Why this project exists,
- Comprehensive API documentation,
- How to contribute (e.g., CONTRIBUTING.md),
    - How to navigate the codebase,
    - Product roadmap,
    - Issues,
    - Idea submissions,
    - Criteria for multimission,
    - Code style expectations,
    - Identify any developer-facing scripts and tools,
    - Testing instructions,
    - Test requirements (e.g., must include complete, automated tests with each pull request, coverage will be checked, etc.),
- Thanks and acknowledgments for the contributors (reward the contributors),
- Where to get help,
- Code of conduct,
- Instructions and template for reporting issues, and
- Licensing (e.g., LICENSE.md).

Additional work to make the open-source projects better and more self-sustaining would include setting up Git Hooks to run tests and perform static code analysis (e.g. code "linting") automatically. The team can even go further by sharing their best integrated development environment (IDE) configuration files to help new developers onboard to the projects easier.

6. Current Progress

The open sourcing effort is progressing well, but it is not without challenges. The main difficulty has been personnel-related issues and not technical. There are NASA flight projects at JPL currently marching toward their launches, and the open sourcing team has had its key, experienced developers reassigned to these flight projects and also to another task in AMMOS that is a higher priority (i.e., command encryption capability implementation). The task team consists of only 1.4 FTE, so these personnel losses are very significant. To mitigate this, new, junior developers were added to the team, but they cannot fully replace the domain knowledge that has been lost and require significant training before they can pick up speed. Despite the challenges, the team is expected to complete the open sourcing task and release the software's source code to the public sometime between June and September 2023.

7. Lessons Learned

Through planning and executing the effort to open source some of the core NASA AMMOS software that have been closed source previously, the NASA AMMOS MCS organization learned a number of lessons that will serve to guide our future technical decisions and approaches. These include: (1) open sourcing new software projects from the beginning should be the new standard approach; (2) always segregate information and data that must be kept private due to export control regulations or their otherwise sensitive nature into one or more separate repositories from the main source repo; (3) be stricter about introducing dependencies—thoroughly analyze their consequences for the open source users; and (4) open sourcing a software creates a new class of users: developers.

By observing how the open sourced software projects such as NASA AMMOS AIT and other NASA software (F' and Open MCT, for example) have experienced greater adoption rates with organizations and users outside their original implementing centers, there is clear evidence that open sourcing is an effective strategy for promoting wider use of NASA software. Because the purpose of NASA AMMOS is to meet the needs of and be used by as many NASA missions as possible—agnostic of the center, partner, or contracting company developing and/or operating them—its individual software products should be open sourced wherever possible. The current task of open sourcing existing mission control software has shown that retroactive open sourcing is both difficult and costly compared to starting the project from the beginning as open source. Going forward, MCS plans to start new projects as open source from the beginning, as long as it would not violate export control regulations.

Another lesson learned is that the MCS development teams should make it a standard practice to keep sensitive information and data, especially those that come under export control, separated from the main software source

repository. With respect to our current effort, if the development teams already had such a policy in place and enforced it, then it certainly would have made the open sourcing effort streamlined and easier. Even if a software is never released as open source, this best practice will allow for stronger access control to restricted information while providing freer access to the rest of the code, data, and information to people in the same organization. Innersourcing, which is a form of open sourcing but limits access to one's own organization, also becomes a possible option more easily with this approach.

When a developer or the team discovers a software library or similar, either online in the public or internally within the organization, and it provides a solution to the problem that the developer or the team is trying to solve, it is all too tempting to introduce it into the software under development as a dependency. In recent years, more rigor has been required to vet those dependencies for possible security vulnerabilities. Widely-known, critical vulnerabilities such as Log4Shell (CVE-2021-44228) and tools being created to help mitigate these kinds of issues (e.g. GitHub Dependabot) should make developers wary of introducing software dependencies without careful consideration. With open sourcing in mind, however, there needs to be another factor in such consideration, which is how a new dependency would impact the users of the software if it ever should be made open source. This anticipatory thinking was lacking when a number of proprietary and JPL/NASA-internal software dependencies were added to AMPCS and other software over the years. Going forward, how a software dependency will impact either the current or possible future open-source users should become a mandatory part of the decision-making analysis.

The final key lesson to share is that, when a software is made open source, it effectively creates a new class of users of the software: developers. For this class of users, the software's API and rest of the code become the product that they consume. Just as a good software product should delight the users that operate the software, a good open-source software project should have APIs and code that are delightful for the developers to read, modify, and extend. As mentioned previously, the core maintainers that are open sourcing the software must not only make available a user's guide, but they must also provide a developer's guide. Ideally, the developer class of users should be treated with equal importance as the regular users of the software. APIs themselves should be treated as first-class citizens in the software project, almost like its own product that must be managed.

Acknowledgements

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

References

[1] Z. Benecken, AMMOS Catalog, Version 5.5, https://ammos.nasa.gov/pdf/AMMOS_Catalog-V5_5_Public_Release.pdf, (accessed 09.02.23).
[2] B. Chess, G. McGraw, Static analysis for security, IEEE Security & Privacy, 2004 Dec 13;2(6):76-9.
[3] Google, Why Open Source?, https://opensource.google/documentation/reference/why, (accessed 09.02.23).
[4] Consultative Committee for Space Data Systems, Space data systems operations with standard formatted data units, Green book, February 1987, https://public.ccsds.org/Pubs/610x0g5.pdf.
[5] R. Ross, V. Pillitteri, K. Dempsey, M. Riddle, G. Guissanie, Protecting controlled unclassified information in nonfederal systems and organizations (2020), https://doi.org/10.6028/NIST.SP.800-171r2.
[6] J. Choi, RESTful CFDP: Managing GDS complexity with microservices, 2018 SpaceOps Conference (p. 2595).