Preliminary Design of the Stellar Apps Software Platform for Developing and Executing On-board Applications

Hendrik Otte [*], Armin Purle-Kopacz [*], Kai Bleeke [*], Arnau Prat [*], Zain Haj Hammadeh [*], Andreas Lund [*], Jan-Gerd Meß [†], Michael Felderer [*‡], and Daniel Lüdtke [*]

[*]*Institute of Software Technology*
German Aerospace Center (DLR)
Braunschweig and Oberpfaffenhofen, Germany

[†]*Institute of Space Systems*
German Aerospace Center (DLR)
Bremen, Germany

[‡]*University of Cologne*
Cologne, Germany

***Abstract*—The German Aerospace Center (DLR) is developing Stellar Apps, an on-board application platform designed for space missions. Third-party applications and system services are deployed as *apps*. These apps can be executed in isolated environments with configurable access to the spacecraft and its computing resources. This allows users to regularly add, replace, remove, and update Apps to improve the flexibility and reusability of multipurpose spacecraft. Complex applications can be executed and tested in a secure environment, reducing their time to orbit. The Stellar Apps platform enables API-based communication not only between different apps, but also with hardware components, allowing it to be integrated into various types of spacecraft, such as satellites, rovers, space stations, and even entire satellite constellations, facilitating data exchange and remote control capabilities through direct communication with the ground station. Access to commonly used artificial intelligence libraries and accelerators paves the way for future space applications. Stellar Apps provides end-to-end services ranging from development on the ground to deployment of Apps in space. The platform is currently available as a prototype at the German Aerospace Center (DLR) and is planned for an in-orbit demonstration during the second year of DLR's CAPTn-1 mission. This paper presents the preliminary design of the software platform.**

***Index Terms*—OBDP, AI, security, software platform, containerization, OBC**

I. Introduction

The Nebulae study [1] concluded that future scientific space missions will rely much more heavily on onboard analysis of scientific data than is currently the case. It will no longer be feasible to transmit all (raw) scientific data back to Earth.

Currently, there is a lot of activity surrounding the development of on-orbit cloud platforms with edge computing capabilities in many areas, such as Earth observation, constellation autonomy, on-orbit servicing, and exploration e.g., [2]–[4]. These missions rely on complex on-board algorithms that increasingly use artificial intelligence. Sometimes these algorithms are so complex that classical verification and validation approaches are not feasible or too expensive. Therefore, new approaches are needed to enable untrusted applications to run on board without compromising the mission.

ESA's OPS-SAT 1 mission [5] is a good example of the concept of allowing untrusted or low technology readiness level (TRL) software to control the satellite. This was achieved by using a secure second on-board computer (OBC) that could take control of the spacecraft in the event of an unsafe situation caused by the experimental software.

The Stellar Apps activity at the German Aerospace Center (DLR), which began in 2025, was motivated by the idea of providing an execution environment for high-performance on-board applications that runs in a safe and secure environment and allows easy access to orbit for low TRL applications, alongside mission-critical systems, such as the *Attitude and Orbit Control System* or the *Command and Data Handling System*. Because of this combination of mixed-criticality applications [6], security and safety are essential aspects of Stellar Apps' design.

The paper first outlines the core directions of related work, followed by an overview of the Stellar Apps' predecessor, ScOSA. It then presents an overview of the platform's use cases and requirements. Next, the preliminary design and the currently available prototype are presented. Finally, the paper concludes with some final remarks and an outlook on future work.

This is the accepted version of the article: H. Otte et al., "Preliminary Design of the Stellar Apps Software Platform for Developing and Executing On-board Applications," 2025 European Data Handling & Data Processing Conference (EDHPC), Elche, Spain, 2025. Conference website: https://atpi.eventsair.com/edhpc-2025/

II. Related Work

As the number of satellites in space continues to rise, and satellites are more capable to perform complex tasks, onboard data processing becomes indispensable. In response to this, the Smart on-board processing for Earth observation systems (SOPHOS) project [7] was initiated to develop technology for high-end data products produced onboard of spacecraft. With a focus on data intensive Synthetic Aperture Radar (SAR) applications, more efficient onboard data processing chains were developed. This includes reduced size of commercial off-the-shelf (COTS) hardware components, improved data compression and processing, as well as hardware utilization optimizations.

The emerging field of artificial intelligence neuromorphic computing holds great potential for advancing space exploration by offering energy-efficient, low-latency, and highly adaptive computing systems [8]. Inspired by the human brain's architecture, neuromorphic computing mimics biological neurons and synapses to build an asynchronous computation unit. Research showed that neuromorphic processors like Intel's *Lohi 1* [9] can minimize the power consumption in space.

Given the need for onboard processing, decision making, and general autonomy, the software running on satellites needs to become more secure and flexible to adapt to future hardware and shifting requirements from the developers.

The NanoSat MO Framework (NMF) [10] represents a pioneering example of middleware that embodies the principles of satellite-as-a-service (SaaS). Notably, the NMF was used in the 2019 ESA OPS-SAT mission. In 2020, a significant milestone was achieved when the NMF was successfully executed in space, enabling external experimenters to upload and run their software as on-board applications. This demonstrated the feasibility of the SaaS concept for "app-driven" satellites. The NMF's architecture provides high-level software abstractions and libraries that facilitate the development of self-contained on-board applications decoupled from the underlying satellite hardware. This design allows developers to create portable, reusable software components. The use of a common Software Development Kit (SDK) and interface across all NMF-enabled on-board units ensures interoperability and consistency, streamlining the development and deployment of on-board applications.

The multiMIND Framework [11], developed with support from the ESA ARTES program, was successfully validated through deployment on the 6U CubeSat EIVE mission, which launched in 2023. The framework's design makes it highly adaptable and multi-mission capable, facilitating the rapid development of diverse payload applications that utilize common application programming interfaces (APIs). Additionally, it allows for the integration of COTS integrated circuit packages (IP cores) within field-programmable gate arrays (FPGAs), accelerating specific tasks such as artificial intelligence or image processing. The multiMIND Framework's ability to provide on-demand software updates and FPGA configurations embodies the principles of SaaS. This capability allows new applications or configuration changes to be deployed efficiently without requiring extensive reimplementation or hardware modifications, enabling more agile and responsive space application development.

III. ScOSA

The predecessor of Stellar Apps at DLR is the Scalable On-board Computing for Space Avionics (ScOSA) Flight Experiment [12]. ScOSA is a distributed, heterogeneous OBC architecture that combines radiation-hardened processors (in this project, a GRE712RC LEON3FT) and multiple commercial off-the-shelf systems-on-a-chip (in this project, eight Xilinx Zynq 7000) in a SpaceWire network. The ScOSA middleware detects failures and migrates tasks from failing nodes to continue execution on another node to achieve high reliability of the system, which is mainly based on radiation-sensitisve COTS components. If the failure was transient, recovered nodes can be automatically reintegrated into the network. The middleware is available for RTEMS and Linux, enabling the development of distributed, high-performance applications. These applications can leverage multiple nodes and utilize the programmable logic on Zynq FPGAs for acceleration purposes.

ScOSA allows multiple applications to be executed concurrently [13]. Applications define a graph of tasks (that processes data) which connected by channels (that store the state of applications). All applications are bundled into a single binary. The middleware maps the tasks and channels of all applications to the different computing nodes. While this allows high system utilization, it comes to the cost of inflexibility during development and a lack of separation between different applications. A fault in a one application can affect the entire system. Developers are further required to design their application around the concept of tasks and channels. Existing software that does not follow this design principle needs to be reimplemented. Stellar Apps, as the successor of ScOSA, aims to improve these shortcomings.

Currently, a 3U-sized, 18-core ScOSA system is being built and integrated into DLR's CAPTn-1 mission, which is scheduled for launch in 2026. This mission will demonstrate the system with five applications in orbit.

IV. Use Cases and Requirements

The Stellar Apps project [14] aims to build on the results of ScOSA. The design of the development and execution environment of Stellar Apps is driven by the lessons learned from the ScOSA activities.

In ScOSA, all applications are compiled into a single binary. This leads to a high degree of dependency between individual applications. A fault in one app can cause the entire collection of applications to fail. Therefore, applications in Stellar Apps are required to run independently from each other. A fault in one application shall not affect the stability of the remaining system. Core system services of the flight software should also be decoupled from each other to increase maintainability and reliability of the system.

Another lesson learned in ScOSA is that application developers would like to have a more flexible development environment with a greater variety of development and runtime libraries. In particular, newer versions of AI environments such as PyTorch or TensorFlow are requested.

Applications are not always planned to be sent to space from the very beginning. Instead, applications often rely on existing code and libraries that were developed without the strict safety requirements for space. Stellar Apps aims to reduce the time to develop applications for space. This includes reusing existing code as much as possible, being able to use existing runtime libraries, and being able to update and test applications *in-situ*.

Nevertheless, running untrusted code in space can cause serious damage to a mission. There is a need for strong separation and isolation of applications. Given such isolation, it would also allow Stellar Apps to provide a testbed for complex algorithms that are not feasible to validate by traditional approaches.

Regular updates of software is required to support an iterative development and software tests in orbit. Application developers want to fix bugs, improve existing features, or introduce entirely new features during a mission. Thus, support for regular software updates during a mission shall be provided by the Stellar Apps platform.

In the Stellar Apps project, seven applications are being developed in parallel with the system software. These applications include Earth observation, on-orbit service image processing, autonomous on-board planning, anomaly detection, an experimental digital twin, and a single event effects detection and mitigation experiment. The applications defined compact use case descriptions to derive requirements for the Stellar Apps software platform, but also for a new reference OBC platform to replace the radiation-hardened node in the ScOSA architecture with a more powerful RISC-V processor. In addition, use cases from industry partners and other use cases such as high performance signal processing are considered and integrated.

Based on the use cases from the application developers, the following three observations are derived:

1) Applications need to collaborate. The services provided by an app shall be accessible to other applications if both developers agree. Otherwise, no data from one app shall be shared with other applications. Stellar Apps can be used as a multi-tenant software platform. Thus, both cooperation and strong separation between tenants need to be supported. Secure communication mechanisms between application are required.
2) Applications can profit from distributed execution on multiple computing nodes. To improve performance, applications can get access to multiple computing nodes. Additionally, different versions of the same application could be executed at the same time. For example, to compare different algorithms in parallel rather than one by one. The Stellar Apps software platform shall therefore support the execution of application distributed on multiple computing nodes.
3) Interactions between the applications and the spacecraft should be simplified. To allow a wide range of app developers creating applications for space systems, an accessible API-based communication with the underlying services and hardware is required. Developers without prior experience in writing software for space, for example from universities or corporations, could launch their ideas into space. Stellar Apps is planned as a platform for a large variety of spacecrafts. Thus, an abstraction layer between applications and the hardware would increase its interoperability.

With an increasing need for on-board autonomy, data-processing, and decision making, applications rely on artificial intelligence more than ever, e.g. [15]–[17]. Stellar Apps shall provide support for hardware accelerators and commonly used runtime libraries for artificial intelligence. In addition, real-time support is also required on the Linux-based nodes to enable real-time applications to meet their timely requirements. The Stellar Apps platform shall therefore provide a real-time capability through integrating real-time co-kernel, namely enabling PREEMPT-RT and the Xenomai 4[1] co-kernel [18]. Co-processor-based solutions, in which real-time tasks are offloaded to a real-time micro-controller such as ARM M4 to run, got less interest as they consume more power and make the platform hardware dependent.

Cybersecurity is another important aspect of Stellar Apps. Satellites are an attractive target for cybersecurity attacks [19]. Therefore, a solid security framework is required to allow multi-tenant operations and to execute untrusted code in space. For example, applications shall not be able to access resources on the spacecraft without explicit permission. Additionally, the amount of computing resources that are available to an app shall be limited. This allows a fair distribution of the resources to multiple applications. It also prevents flawed applications from exceeding the available resource pool, which could lead to a total failure of the mission. Constant monitoring in the form of intrusion detection, common weakness,and common vulnerability analysis before deployment are necessary.

To summarize, the following key requirements are identified:

1) Isolation and separation of individual applications
2) Regular updates of applications
3) Strong cybersecurity measurements during the entire lifecycle of an application
4) Multi-tenant operations
5) Support for real time applications
6) API-based communication between applications and the underlying hardware
7) Access to AI accelerators and commonly used runtime libraries
8) Distributed execution of applications on multiple computing nodes

[1]https://v4.xenomai.org/overview/

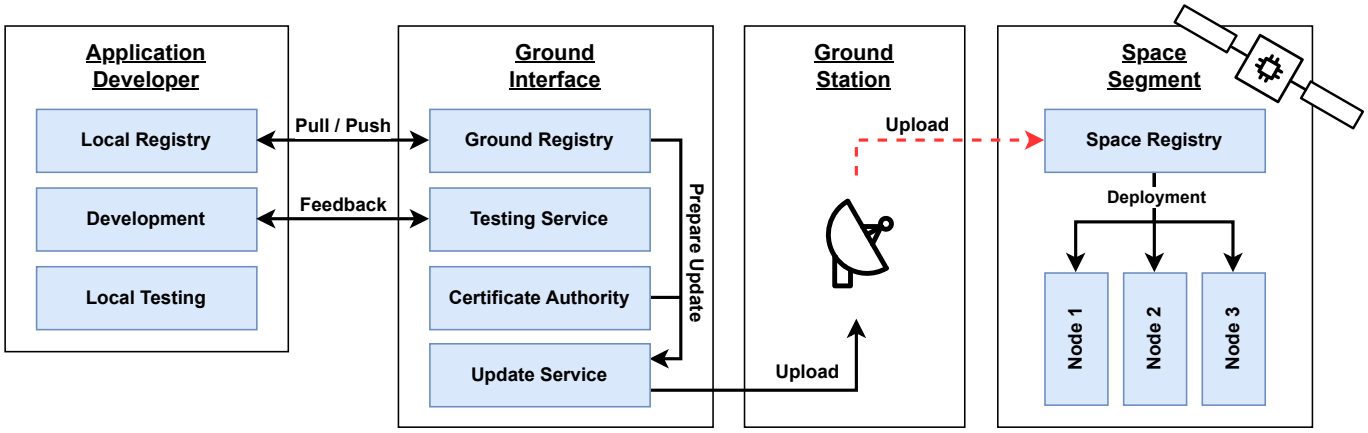

Fig. 1. There are three main components of Stellar Apps: 1) the application developer; 2) the trusted ground interface; and 3) the space segment. Developers use the ground interface to create apps. These apps are then submitted to the ground interface for further security checks and verification. Finally, the apps are sent into space, where they are executed on the spacecraft.

V. Preliminary Design of the Stellar Apps Software Platform

The Stellar Apps software platform has been developed based on the previously described use cases and requirements. Stellar Apps is built on the foundations of the ScOSA project described in Sect. III. Stellar Apps is organised into three main components: the space segment, the ground interface and the application development. These components work together to enable the secure execution of potentially untrusted third-party applications in space (Fig. 1).

The Stellar Apps ground interface is a trusted institution that connects application developers with the spacecraft. The ground interface provides a container-based development environment. Application developers can use to develop and test Stellar Apps. These applications are then submitted to the ground interface for further checks and verification before being uploaded to the satellite. Once uploaded, the applications are deployed and executed by the Stellar Apps software platform on the space segment.

A. Space Segment

The Stellar Apps software platform is primarily responsible for securely and safely executing uplinked applications alongside mission-critical services. Fig. 2 illustrates the software layers that are run by each computing node.

The operating system is Linux, which has been extended to support real-time applications. The real-time PREEMPT-RT Linux kernel is considered to support real-time applications. In addition, the co-kernel Xenomai 4 is integrated to support hard real-time requirements. The flight software core provides essential services to the system and monitors application execution.

Applications and libraries are deployed in the form of containers. This allows individual applications to be isolated from each other. The amount of computing resources can be limited and access to certain files or hardware can be restricted. If an application exhibits undesirable behaviour, such as crashing or excessive memory usage, the impact on the spacecraft is limited.

On top of the operating system and network layers, the communication layer controls the data exchange between individual applications (Fig. 2). By default, no data can be shared between apps. The communication layer only allows information to be exchanged with explicit permission from all participating applications. Stellar Apps supports the distributed execution of applications on multiple computing nodes as illustrated in Fig. 3. Therefore, the communication layer also handles messages between applications on different nodes.

An application orchestrator is responsible for starting, stopping, installing, updating, and removing applications from the platform. It controls the containerization engine, which isolates applications and limits their access to the system. If non-nominal behavior is observed, the application orchestrator can terminate applications. If a malfunctioning application is detected, a detailed report is sent to the ground, and the application is stopped.

A registry stores applications and provides a set of supported frameworks and libraries in different versions. In contrast to the ScOSA Flight Experiment, it is no longer necessary for all applications to rely on the same library version. With this approach, different versions of the application can be deployed and run.

Applications and libraries in the registry can be updated. The ground interface sends a patch file to the spacecraft. The patch's authenticity can be verified by its signature. Then, the patch is applied, and the new version is stored in the registry and made available for deployment. Previous versions of an app can be kept as a backup option in case the patch becomes corrupted.

For artificial intelligence (AI) applications, at least two frameworks are supported: PyTorch and TensorFlow. In addition to the FPGA co-processor architecture of ScOSA, the integration of special AI FPGA designs [20] and the integration of embedded GPUs is planned.

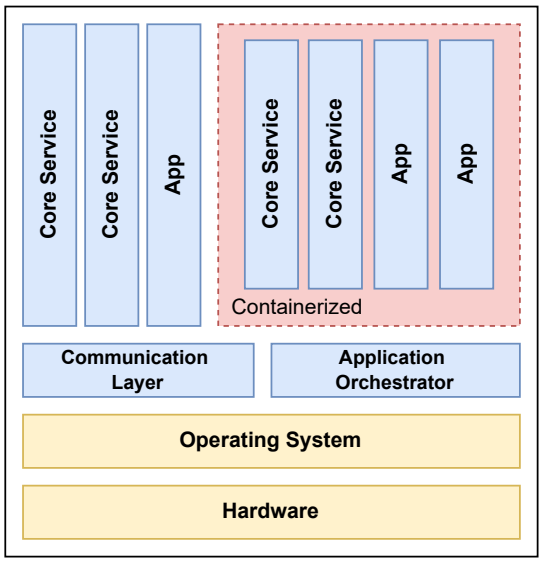

Fig. 2. Stellar Apps' layered software stack is designed for a single node. A communication layer handles secure communication between applications. The application orchestrator manages application installations, updates, removals, and activation. The entire system runs on a Linux operating system.

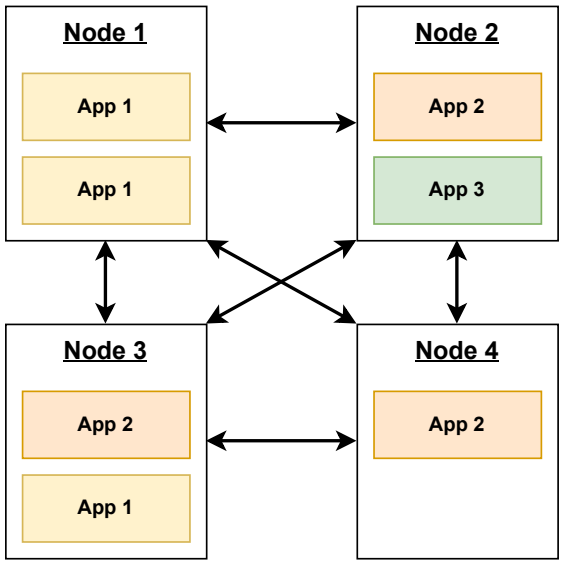

Fig. 3. Applications can be executed on a single computing node, or spread over multiple ones. Communication channels between the nodes allow data transfer between apps. Data can only be shared if applications agree.

For the interfaces to sensors and actuators, to other applications on the spacecraft and to the ground, APIs are provided that abstract from the actual communication protocol. This simplifies application development by eliminating the need to implement the details of specific telecommand and telemetry service standards. The programming model of Stellar Apps is similar to the programming of lightweight web services.

While Stellar *Apps* do not necessarily have to be implemented in a particular programming language, APIs for C/C++ and Rust will be provided. Rust was chosen because of the safe-by-design nature of the language specification and the promising results of a Rust study for ESA [21]. Python will also be supported because of its widespread use in the AI community. However, we will not provide a direct Stellar Apps Python API in the first phase.

The Stellar Apps environment will be provided at least for the GR712 and Xilinx Zynq 7000 (i.e. ScOSA on the CAPTn-1 mission), Intel x64, and the new RISC-V-based reference platform. For the network layer, both SpaceWire and TSN are supported.

B. Ground Interface

To execute multi-tenant applications with varying sets of permissions, a trusted interface between the application developers and the execution in orbit is required. Therefore, the Stellar Apps ground interface is introduced. It is a collections of various services, developers, and infrastructure to provide services to the application developers and to operate the spacecraft. The main purpose of the ground interface is to verify the authenticity of applications and their permissions on the spacecraft. The ground interface provides the following functionalities:

1) **Provide a development environment to the application developers.** Application developers shall be able to test their applications independently from other developers. The Stellar Apps ground interface provides a container-based environment that allows local and independent software tests. This enables developers to run their applications in a similar software environment that is present on the spacecraft.
2) **Provide testing capabilities to the application developers.** Not every developer has access to the specialized hardware that is used on the spacecraft. Therefore, the ground interface provides services to the developers to test their applications on such hardware.
3) **Analyse the submitted apps for security issues.** Applications that are submitted to the ground interface are scanned for security issues with both static and dynamic analysis methods. The applications are also checked against known vulnerabilities and malicious code.
4) **Negotiate and verify the entitlements that an application developer has access to.** By default, an application has no permissions to run on a spacecraft. Accessing hardware resources, software services on the spacecraft, or communicating to other apps requires explicit authorization. The ground interface provides entitlements to the developers. Each entitlement grants a set of permissions on the spacecraft. As a certificate authority, the ground interface also proves the correctness and authenticity of these entitlements to the spacecraft.
5) **Upload and update applications on the spacecraft.** To upload applications, developers submit their apps to the ground interface. The applications go through extended testing and verification. The ground interface then creates an app bundle with additional information

about granted entitlements and other app specific details. The app bundle is send to the spacecraft using an existing ground station infrastructure.

Applications are deployed in the form of *app bundles*. An app bundle contains everything that is required to execute an application. This includes the container image, a set of granted entitlements, the developer identifier, information about the execution mode, and signatures from the ground interface to prove authenticity of the bundle.

C. Application Development

Most functionalities for the application developers are covered by the ground interface described in Sect. V-B. Developers can access a container-based environment to develop and test their applications. Applications are submitted to the ground interface for hardware tests and verification. Once the application is ready, the developer requests the ground interface to upload the app to the spacecraft.

In order to access certain software or hardware systems on the spacecraft, developers need to be granted the corresponding entitlements. Whether an entitlement is granted or not is a decision made by the operators of the ground interface. Given the potential risks to a mission, granting an entitlement should be carefully considered.

VI. Prototype

A Stellar Apps prototype is developed by DLR as a proof-of-concept and platform to validate design decisions and conduct experiments. The following section gives a brief overview of the current state of this prototype.

Stellar Apps is conceptualized as an ecosystem for the secure and safe execution of third-party Apps in space. This includes software in space but also infrastructure on ground. Stellar Apps is a collection of different software products that are used by application developers, service providers on ground, and spacecrafts. Therefore, the prototype is also made up of multiple individual software components for all three segments of Stellar Apps.

Fig. 4 illustrates the components included in the prototype:

- Command-line interface for the application developers.
- Server for the Ground Interface to provide the development environment and to receive Apps.
- Command-line interface for the Ground Interface to manage and upload Apps to the spacecraft.
- Registry servers to store and manage Apps. For both the Ground Interface and the spacecraft.
- Application Manager for the spacecraft.

All components are developed with Rust. The communication between the components is realized with TCP/IP. The prototype is split into three dedicated virtual machines to reflect the division between the three segments of Stellar Apps: The developer, the ground interface, and the spacecraft.

Following the design of Stellar Apps, individual applications are deployed in the form of containers. A container image is a portable and executable package of a software application and its dependencies. It is used to create containers that provide a consistent and reliable environment for the application to operate in. The OCI[2] (Open Container Initiative) standard for container images is used. Its wide spread use in containerization tools makes it compatible with various existing containerization solutions like Docker[3] or Podman[4].

Registries are used to manage the available container images on a system. A registry is a piece of software to store, version, update and remove container images. Such a registry is found on all three segments of Stellar Apps. The developers have access to a registry with the images that were downloaded from the ground interface. These images provide the development environment for Stellar Apps. The developed apps are then pushed back into the registry of the ground interface. From there, apps are uploaded into the local registry on the spacecraft from where they are deployed and executed as presented in Fig. 4.

To manage container images received from the ground interface, developers have access to a CLI (command-line interface). This interface is used to request different development environments from the ground interface, and to build, run, test, and upload apps. A similar CLI is used on the ground interface. The CLI can be used to manage applications and upload them to the spacecraft. The spacecraft receives apps in the form of application bundles that are stored and managed in its local registry. Finally, the application orchestrator deploys and executes the available container images on the spacecraft.

The application orchestrator also provides an interface to the environmental simulator *42* [22], which is used in the *NASA Operational Simulator for Small Satellites* (NOS3) [23]. It is used as a source for the precise simulated position of the spacecraft and to test how to operate the spacecraft from ground. This process is illustrated in Fig. 5.

The prototype is being tested on a set of multiple *Raspberry Pi 4 Model B*. Each Raspberry Pi represents a single computing node. Multiple nodes can communicate using IP-based communication channels to test and validate the system design and the implemented distributed computing algorithms.

VII. Conclusions and Outlook

With the Stellar Apps platform a secure and safe environment for on-board applications is currently developed that will significantly reduce the time-to-orbit for even experimental applications. The software platform is targeted for a large variety of spacecraft for future space missions ranging from, for instance, satellites, space transportation, rovers, or experiments on space stations. It enables edge computation by providing a software environment for high-performance OBCs with the support of hardware acceleration for AI applications. Stellar Apps is designed to be compatible with a wide range of hardware architectures and provides an powerful development environment based on the Linux operating system. Additionally, the Stellar Apps platform is planned to be extended to

[2]https://opencontainers.org/ (Accessed September 14, 2025)
[3]https://www.docker.com/ (Accessed September 14, 2025)
[4]https://podman.io/ (Accessed September 14, 2025)

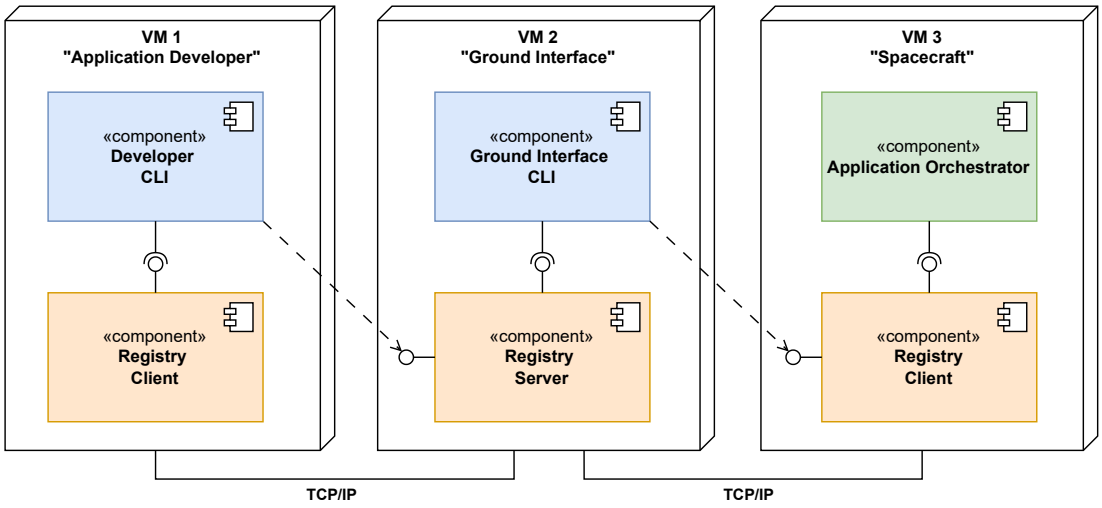

Fig. 4. Overview of the prototype components and how they relate to each other. The three segments of Stellar Apps (Developer, Ground Interface, Spacecraft) can be deployed into distinct (virtual) machines. Each machine has access to a registry for container images. Command-line interfaces are used to manage and upload the images. An application manager on the spacecraft machine is responsible for deploying the applications. TCP/IP connections are used to share data between the machines.

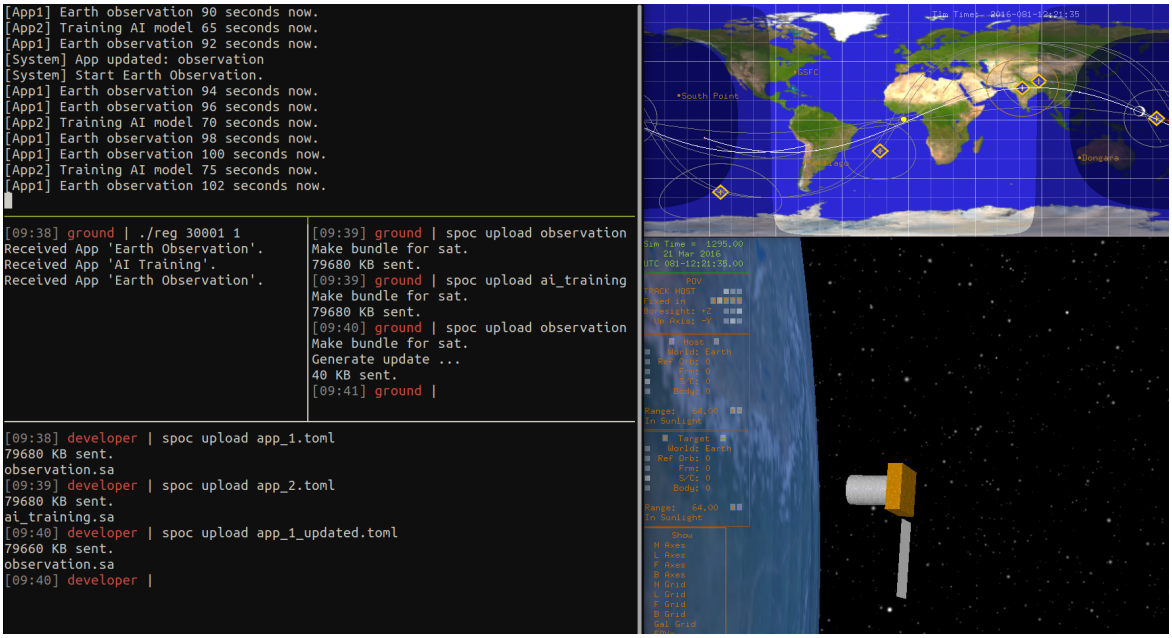

Fig. 5. A screenshot of the prototype that shows 1) an interactive dashboard on the left, and 2) a connected instance of 42 - Spacecraft Simulator on the right. The dashboard is showing the three segments of Stellar Apps: The application developer (bottom), the ground interface (center), and the spacecraft (top). The instance of the 42 - Spacecraft Simulator is used for visualization and simulation of the environment.

support multi-spacecraft collaborations, for example in the context of satellite constellations using inter-satellite links.

Stellar Apps is under development in a three year project. First results with relevant applications in orbit are demonstrated in the second mission year of CAPTn-1. In parallel ground tests will be conducted with the new reference platform OBC.

References

[1] J. V. Hook, J. Castillo-Rogez, R. Doyle, T. S. Vaquero, T. M. Hare, R. L. Kirk, V. Fox, D. Bekker, and A. Cocoros, "Nebulae: A proposed concept of operation for deep space computing clouds," in *2020 IEEE Aerospace Conference*, 2020, pp. 1–14.

[2] M. Der Yang, H. H. Tseng, Y. C. Hsu, and W. C. Tseng, "Real-time crop classification using edge computing and deep learning," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2020, pp. 1–4.

[3] B. Denby and B. Lucia, "Orbital edge computing: Nanosatellite constellations as a new class of computer system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 939–954.

[4] I. Leyva-Mayorga, M. Martinez-Gost, M. Moretti, A. Pérez-Neira, M. Á. Vázquez, P. Popovski, and B. Soret, "Satellite edge computing for real-time and very-high resolution earth observation," *IEEE Transactions on Communications*, vol. 71, no. 10, pp. 6180–6194, 2023.

[5] D. Evans and M. Merri, "OPS-SAT: A ESA nanosatellite for accelerating innovation in satellite control," in *SpaceOps 2014 Conference*, 2014, p. 1702.

[6] A. Burns and R. Davis, "Mixed criticality systems-a review," *Department of Computer Science, University of York, Tech. Rep*, 2022.

[7] O. Flordal, N. Gollin, M. Jaeger, V. Kollias, M. Martone, J. Naghmouchi, M. Persson, S. Pedersen, N. Pogkas, R. Scheiber *et al.*, "Smart on-board processing for next generation sar payloads," in *EUSAR 2024; 15th European Conference on Synthetic Aperture Radar*. VDE, 2024, pp. 606–610.

[8] A. Yu, S. Woo, and H. Ahn, "Toward transforming space exploration with artificial intelligence neuromorphic computing," *Engineering Applications of Artificial Intelligence*, vol. 154, p. 111055, 2025.

[9] M. S. Murbach, E. Barszcz, L. S. Schisler, A. J. Salas, K. Boateng, G. Marty, M. Mooney-Rivkin, A. Brock, S. M. Krześniak, and S. Zuniga, "Brainstack–a platform for artificial intelligence & machine learning collaborative experiments on a nano-satellite," in *SmallSat Conference Proceedings*, no. SSC23-X-03. SmallSat Organization, 2023.

[10] C. Coelho, O. Koudelka, and M. Merri, "Nanosat mo framework: When obsw turns into apps," in *2017 IEEE Aerospace Conference*. IEEE, 2017, pp. 1–8.

[11] A. Pawlitzki and F. Steinmetz, "multimind–high performance processing system for robust newspace payloads," in *2nd European Workshop on On-Board Data Processing (OBDP2021)*, 2021.

[12] D. Lüdtke, T. Firchau, C. E. Gonzalez Cortes, A. Lund, A. M. Nepal, M. M. H. H. Elbarrawy, Z. A. Haj Hammadeh, J.-G. Meß, P. Kenny, F. Brömer, M. Mirzaagha, G. Saleip, H. Kirstein, C. Kirchhefer, and A. Gerndt, "Scosa on the way to orbit: Reconfigurable high-performance computing for spacecraft," in *Proceedings - 2023 IEEE Space Computing Conference, SCC 2023*, August 2023. [Online]. Available: https://elib.dlr.de/196642/

[13] A. Lund, Z. A. Haj Hammadeh, P. Kenny, V. Vishav, A. Kovalov, H. Watolla, A. Gerndt, and D. Lüdtke, "Scosa system software: the reliable and scalable middleware for a heterogeneous and distributed on-board computer architecture," *CEAS Space Journal*, vol. 14, no. 1, pp. 161–171, 2022.

[14] D. Lüdtke, J. Sommer, C. Gonzalez, H. Otte, A. Lund, Z. H. Hammadeh, C. Brokering, and A. Prat, "Stellar apps: On-board application framework for space missions," 2025, 18th Annual Flight Software Workshop.

[15] G. Labrèche, D. Evans, D. Marszk, T. Mladenov, V. Shiradhonkar, T. Soto, and V. Zelenevskiy, "Ops-sat spacecraft autonomy with tensorflow lite, unsupervised learning, and online machine learning," *2022 IEEE Aerospace Conference (AERO)*, pp. 1–17, 2022.

[16] S. Kacker, A. Meredith, K. Cahoy, and G. Labreche, "Machine learning image processing algorithms onboard ops-sat," in *Proceedings of the AIAA/USU Conference on Small Satellites*, 2022, in Science/Mission Payloads, SSC22-WKIV-03.

[17] G. Labrèche, C. Guzman, and S. Bammens, "Generative ai... in space! adversarial networks to denoise images onboard the ops-sat-1 spacecraft," *2024 IEEE Aerospace Conference*, pp. 1–17, 2024.

[18] J. H. Brown and B. Martin, "How fast is fast enough? choosing between xenomai and linux for real-time applications," in *proc. of the 12th Real-Time Linux Workshop (RTLWS'12)*, 2010, pp. 1–17.

[19] J. Pavur and I. Martinovic, "Building a launchpad for satellite cybersecurity research: lessons from 60 years of spaceflight," *Journal of Cybersecurity*, vol. 8, no. 1, p. tyac008, 2022.

[20] D. Helms, Q. Dariol, K. Grüttner, B. R. Perjikolaei, L. Einhaus, and S. Gregor, "Fpga based in-memory ai computing," in *ONERA Workshop on Advances in Artificial Intelligence for Aerospace Engineering*, Mai 2023. [Online]. Available: https://elib.dlr.de/194973/

[21] J. Sommer, T. Gutierrez Rojo, A. Lund, H. I. E. Abdelmaksoud, and D. Lüdtke, "Viability of rust for avionics software development – current status and way forward," in *7th Workshop on Avionics Systems and Software Engineering*, ser. SE 2025 - Companion Proceedings. Gesellschaft für Informatik, Februar 2025. [Online]. Available: https://elib.dlr.de/212971/

[22] E. T. Stoneking, "42 – Spacecraft Simulation," Accessed: September 14, 2025. [Online]. Available: https://github.com/ericstoneking/42

[23] NASA, "NASA Operational Simulator for Small Satellites (NOS3)," Accessed: September 14, 2025. [Online]. Available: https://github.com/nasa/nos3