18th International Conference on Space Operations, Montreal, Canada, 26 - 30 May 2025.

SpaceOps-2025, ID # 390

A Scalable Architecture for Integrating AI and Non-AI Agents in Space Mission Control Applications

José Feiteirinha[a]*, Diogo Vicente[b], Luigi Palladino[c],

[a] *VisionSpace Technologies GmbH, jose.feiteirinha@external.eumetsat.int*
[b] *European Organisation for the Exploitation of Meteorological Satellites, diogo.vicente@eumetsat.int*
[c] *European Organisation for the Exploitation of Meteorological Satellites, luigi.palladino@eumetsat.int*
* Corresponding Author

Abstract

EUMETSAT is an intergovernmental organisation based in Darmstadt, Germany, with 30 Member states that operates fleets of satellites for monitoring the weather, climate and the environment from space. With long duration satellite programmes like MSG or EPS, and respective follow-ons MTG and EPS-SG, EUMETSAT will continue to deliver data and products vital to weather forecasting and contributes significantly to the monitoring of the weather and climate change. To ensure efficiency and effectiveness, EUMETSAT endeavours to establish Multi-Mission Ground System approaches, where common solutions are deployed across multiple programs with minimal customisation. Artificial Intelligence (AI) has demonstrated transformative potential across various domains, yet its application in mission control applications remains an area of active exploration. This paper presents a scalable architecture for incorporating AI capabilities into Monitoring and Control (M&C) systems, particularly focusing on legacy applications not originally designed for AI integration. The architecture integrates both AI-powered and traditional non-AI components, accommodating diverse elements such as simulators, control systems, and procedure management tools within a unified framework. By leveraging Large Language Models (LLMs) and natural language processing, the system enhances automation capabilities while maintaining compatibility with established systems. The presence of legacy systems incompatible with standard AI agent architectures presented a key challenge, leading to the development of a bespoke solution. Our evaluation of existing AI agent architectures revealed none fully met our requirements, prompting the design of a novel approach. The rapidly evolving landscape of AI and LLMs added complexity, necessitating a flexible, adaptable architecture. The developed framework enables gradual integration of AI capabilities without disrupting critical procedures. Through natural language interfaces powered by LLMs, we've created an intuitive system where AI agents can interact with non-AI agents—for example, controlling spacecraft simulation scenarios through natural language commands. The architecture's adaptability to different agent types, from full AI-agents to traditional simulation modules, ensures evolution of AI capabilities while maintaining operational reliability. Initial testing has demonstrated the architecture's potential for enhancing test and validation processes, providing a pragmatic solution for organizations seeking to integrate AI capabilities into existing M&C systems without requiring complete system overhauls. By enabling the coexistence of AI and non-AI components, it allows for incremental adoption of AI technologies while minimizing operational disruption. This work serves as an enabler for AI in space mission control applications, reducing technical barriers and empowering teams to reimagine operations enhanced by AI capabilities. The architecture's focus on compatibility, scalability, and adaptability creates a framework for strategic incorporation of AI technologies into legacy systems, while its flexible design accommodates future advancements in both AI capabilities and operational requirements.

Keywords: Artificial Intelligence, Mission Control Systems, EUMETSAT, Large Language Models, Legacy Systems, Agent Architecture, Maintenance Lifecycle

Acronyms/Abbreviations

AI Artificial Intelligence
APT Anomaly Processing Tool
AR Anomaly Report
API Application Programming Interface
CO2M Copernicus Carbon Dioxide Monitoring
CORBA Common Object Request Broker Architecture (legacy protocol in SCOS-2000)
EGOSCC European Ground Operation System Common Core
EUMETSAT European Organization for the Exploitation of Meteorological Satellites
EPS EUMETSAT Polar System
EPS-SG EUMETSAT Polar System Second Generation
FARC File Archive
FOIMS Flight Operations Information Management System
GFTS Generic File Transfer System
GPU Graphics Processing Unit
LLM Large Language Model
MCAT Mission Control Application and Tools
MCS Mission Control System
MISC Miscellaneous (SCOS-2000 dynamic server)
MOF Mission Operations Facility
MPS Mission Planning System
MTG Meteosat Third Generation
MSG Meteosat Second Generation
NLP Natural Language Processing (integral to LLM usage)
RAG Retrieval-Augmented Generation
SATSIM Satellite Simulator
SCOS Spacecraft Control and Operations System
S6 Sentinel-6
SLE Space Link Extension
TC Telecommand
TM Telemetry
VC Virtual Channel

1. Introduction

Space mission control applications are among the most critical and complex systems in modern aerospace operations, tasked with ensuring the safe and reliable management of spacecraft. At organizations like the European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT) and the European Space Agency (ESA), these mission control systems (MCS) have evolved over decades to meet the demands of increasingly sophisticated missions.

EUMETSAT is an intergovernmental organisation responsible for establishing, maintaining and exploiting satellite systems for monitoring the weather, the climate and the environment from space. It operates fleets of Geostationary satellites like MSG or MTG and Polar-orbiting satellites like EPS and EPS-SG. In addition, EUMETSAT is entrusted by the European Union to operate and exploit 4 Sentinel Missions part of the Copernicus Programme: Sentinel-3, Sentinel-4, Sentinel-5, Sentinel-6). EUMETSAT's Mission Control Centres are located in Darmstadt, Germany and provide the dedicated Flight Control teams with relevant Mission Control functions required to the monitor and control of the different satellites. EUMETSAT promotes the adoption of Multi-Mission Ground System approaches by deploying common solutions across multiple programmes that require minimal customization/extensions as a way to exploit synergies across programmes.

Despite the transformative potential of artificial intelligence (AI) in various domains, its adoption within MCS—both operationally and for maintenance—has been virtually non-existent[1]. While AI has found applications in data processing and review[2], the high stakes, rigorous safety requirements, and entrenched legacy infrastructure of MCS have made its integration either too risky or too troublesome, with no guaranteed success until recently.

The maintenance lifecycle of these systems exemplifies the challenges that have hindered AI adoption. Engineers working on anomaly investigation, procedure validation, and system testing face steep learning curves due to the complexity of legacy systems like SCOS-2000 and SIMULUS. Time constraints often result in poor documentation of anomalies, complicating future troubleshooting efforts, while non-reproducible anomalies can linger unresolved for months as teams struggle to recreate triggering conditions. These inefficiencies are compounded by the mission-critical nature of space operations, where even minor changes require exhaustive validation to prevent potential impacts on live missions. Traditional approaches to maintenance have thus remained manual and expertise-intensive, lacking the automation that AI could provide.

Increased automation via AI-based systems can significantly reduce operational loads on teams, minimize human error, and enhance anomaly detection accuracy, yielding substantial benefits over time. This is particularly relevant for long duration satellite programmes such as MTG and EPS-SG, with planned operational lifetime beyond 20 years. This paper introduces a groundbreaking shift: a scalable architecture that integrates AI capabilities into MCS maintenance workflows without compromising operational integrity. Recognizing the historical reluctance to adopt AI in this domain, our solution leverages Large Language Models (LLMs) to enhance automation in a risk-free manner. Designed with compatibility in mind, it seamlessly integrates with legacy systems, addressing a key barrier to AI adoption. By focusing on maintenance tasks—such as automating anomaly reproduction and procedure validation—we provide a practical entry point for AI that avoids the risks associated with real-time mission control, offering measurable efficiency gains while preserving the safety and reliability that EUMETSAT and ESA prioritize.

What sets this architecture apart is its replicability, making it a model that other space agencies and organizations can adopt. Initial testing demonstrates its potential to reduce anomaly reproduction time from days to hours, alleviating the burden on maintenance teams and improving documentation quality. This work not only bridges the gap between modern AI capabilities and legacy MCS but also establishes a foundation for incremental AI adoption across the space operations community.

2. Background

European space operations have evolved through multiple generations of control systems, each building upon lessons learned while adhering to stringent requirements for mission safety and reliability. The current operational landscape is dominated by two major system families, each encompassing core control capabilities and a suite of associated support tools. These systems, developed and refined over decades, underpin the complex task of managing spacecraft for organizations like the European Organisation for the Exploitation of Meteorological Satellites (EUMETSAT) and the European Space Agency (ESA). At EUMETSAT maintenance activities tasks, such as anomaly investigation, procedure validation, and system testing highlight a critical yet often overlooked aspect of space operations, which we explore in greater detail in Section 2.2. Together, these operational and maintenance processes form a robust ecosystem designed to ensure mission success, balancing legacy infrastructure with the demands of increasingly sophisticated missions.

2.1. EUMETSAT Space Operations Systems

To harmonise and streamline mission control solutions, many of the roadmaps include alignments across missions, and this has led to the creation of C-MCS and C-SATSIM which are mission control and satellite simulation based on ESA/MICONYS [3] and ESA/SIMULUS.

2.1.1. MICONYS ecosystem

The Mission Control System Infrastructure (MICONYS) represents the current mission control solution used as baseline for different ESA and EUMETSAT missions. At its core is SCOS-2000 (Spacecraft Control and Operations System), which provides fundamental monitoring and control capabilities. However, MICONYS extends beyond basic control functions to include a comprehensive suite of tools:

- SCOS-2000: The central mission control system component
- Network Interface System (NIS) for ground station communications
- Space Link Extension (SLE) API for standardized space communication
- Engineering Data Distribution System (EDDS) for telemetry processing

- Mission Planning System (MPS) for operations scheduling
- Generic File Transfer System (GFTS) for data file handling

2.1.2. SIMULUS ecosystem

Similarly to MICONYS, SIMULUS represents the baseline used for the development and instantiation of mission specific operational satellite simulators used in different ESA and EUMETSAT missions. These systems are used for different activities such as: development of operational procedures, operator training, ground segment verification and validation.

2.1.3. EGS-CC

The European Ground Systems Common Core (EGS-CC)[5] is an European initiative to develop a common infrastructure to support the development of ground space systems for space missions. This ecosystem will likely lead to relevant monitoring and control solutions that can be used as baseline for future missions.

2.1.4. Others

Beyond the core platforms of MICONYS, SIMULUS, and the emerging EGS-CC, a variety of additional systems support European space operations, each contributing specialized functionality to the mission control ecosystem. These systems range from fully integrated components within established platforms to standalone tools tailored to specific operational needs. For instance, the Mission Planning System (MPS) and Flight Operations Management Tool are critical in planning and documenting operational procedures, yet their integration varies. Some MPS implementations, such as those managing scheduling for EUMETSAT missions, operate independently, interfacing with core systems like SCOS-2000 through custom workflows rather than being embedded within the MICONYS suite. Similarly, the Procedure Management Tool supports procedure development and validation but often exists as a standalone tool, bridging operational and maintenance teams without direct dependency on a single platform.

This diversity reflects the heterogeneous nature of space operations, where legacy systems, mission-specific requirements, and evolving standards coexist. While some tools are deeply embedded within platforms like MICONYS or SIMULUS—enhancing their functionality—others remain peripheral, offering flexibility at the cost of tighter integration. Understanding these relationships is essential for assessing the challenges and opportunities of AI integration, particularly as we explore maintenance-focused applications in subsequent sections.

2.2. Maintenance lifecycle

Given the critical nature of space operations and the challenges of integrating AI into operational workflows, we identified the maintenance lifecycle as an optimal entry point for AI integration. Maintenance activities operate one step removed from real-time mission operations, creating a safer environment for innovation while addressing significant operational challenges. Activities range from corrective maintenance, such as anomaly investigation to fix existing issues; preventive maintenance, like procedure validation to avert future problems; adaptive maintenance, which involves updating systems to meet new mission requirements; and perfective maintenance, aimed at enhancing system efficiency or performance. Collectively, preventive, adaptive, and perfective efforts represent non-corrective maintenance, focusing on proactive improvement rather than reactive fixes. For example, when a maintenance engineer investigates an anomaly (corrective maintenance), work includes the usage of satellite simulators and test environments instead of live spacecraft systems. Similarly, validating procedures (preventive maintenance) ensures operational reliability, adapting systems to new conditions (adaptive maintenance) prepares for future challenges, and optimizing system performance (perfective maintenance) boosts efficiency. This isolation from live operations provides a natural safety barrier, enabling the integration of AI to enhance not just corrective tasks but also non-corrective activities—predicting issues, adapting to changes, and refining systems—without risking spacecraft operations. Furthermore, maintenance activities often suffer from inefficiencies that AI could probably address. Engineers frequently spend hours or days reproducing reported anomalies, searching through scattered documentation, or validating procedures across different mission configurations. These tasks require significant expertise but are often bottlenecked by manual processes and fragmented information. By focusing our AI integration efforts on the maintenance lifecycle, we can demonstrate tangible benefits while building confidence in AI technologies for space operations. This approach also

allows us to develop and refine our agent architecture in a demanding but lower-risk environment, creating a foundation for potential future expansion into other operational areas.

2.3. The rate of change in AI and implications for this work

In the fast-evolving landscape of artificial intelligence (AI), the pace of development presents both opportunities and challenges for space mission control systems. If we fail to adapt, we risk obsolescence as AI capabilities outstrip our technologies. Conversely, if we set overly ambitious expectations—assuming AI will soon resolve all complexities of our systems—and take no action, we may miss out on near-term benefits. For this reason, in the work performed we aimed at striking a balance between pragmatism and proactivity. As depicted in Figure 1, we aimed at adjusting existing technology stacks to be compatible with AI advancements that we can anticipate.

Our architecture leverages REST APIs as standardized interfaces for AI agents to interact with these systems. By abstracting the underlying complexity, REST APIs allow AI to perform critical tasks—such as retrieving telemetry data, configuring simulations, or triggering operational sequences—without requiring a detailed understanding of the systems' user interfaces or internal workings. For instance, rather than an AI agent attempting to manipulate a simulator's graphical interface, it can issue a simple REST API call (e.g., `POST /simulation/run` with a JSON payload specifying parameters) to achieve the same result efficiently. This approach significantly reduces the computational and developmental overhead that direct manipulation technologies like Computer Use would entail.

As shown in the picture above, we propose a balanced approach to AI integration in space mission control. Through REST APIs, we bridge the gap between today's complex systems and tomorrow's AI potential, ensuring that we can leverage its benefits effectively and efficiently in the medium to short term. This preparation positions us to stay ahead in an AI-driven era, delivering tangible value to space mission operations without waiting for speculative breakthroughs.

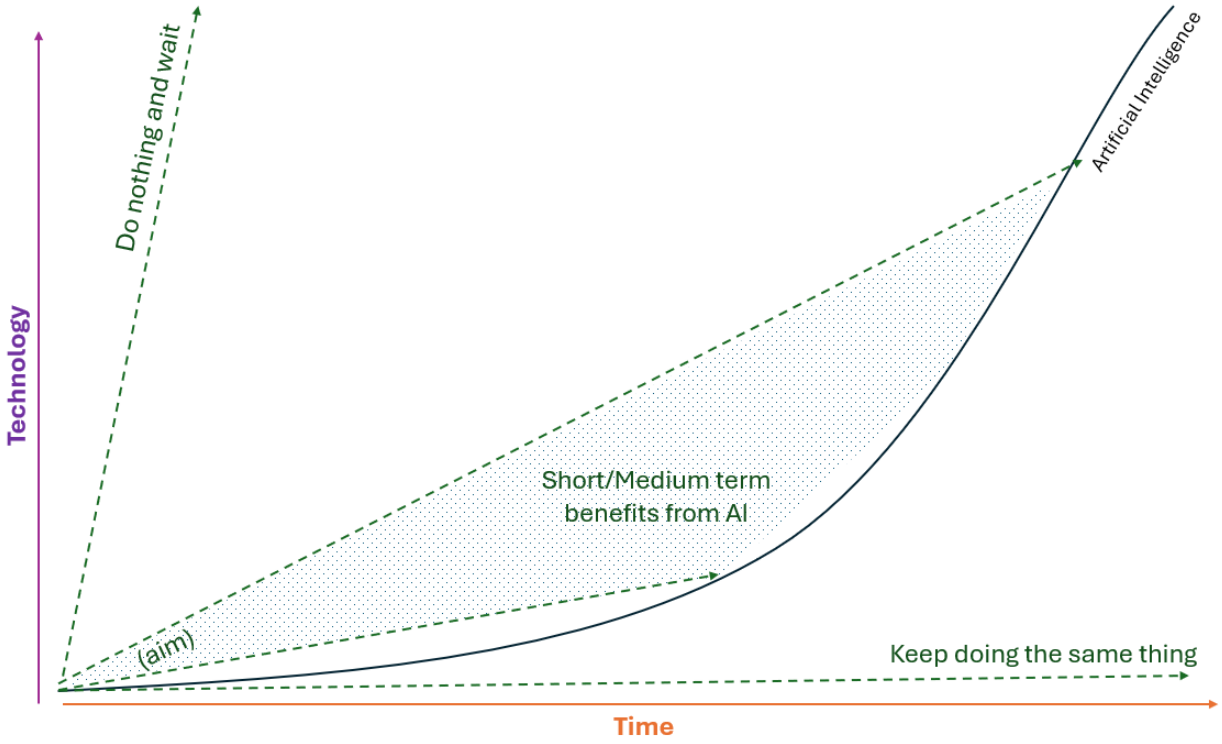

Fig. 1. Diagram showing aim strategy

2.4. Integration Challenges

Both legacy and new generations of systems share characteristics that make AI integration particularly challenging:

The mission-critical nature of these systems requires exhaustive validation of any changes. Even minor modifications must undergo rigorous testing to ensure they cannot impact operational safety. This creates a significant barrier to introducing AI capabilities, as machine learning models can be difficult to verify to the same standards as traditional software.

The interfaces between system components have evolved over decades of operations. Many of these interfaces use specialized protocols (like CORBA in SCOS-2000) or complex data formats optimized for space operations. This legacy creates significant challenges for modern AI tools, which typically expect standardized web protocols and data formats.

Operational procedures and safety requirements have been refined through years of mission experience. These procedures often assume human decision-making at critical points, making automation through AI a complex challenge requiring careful consideration of failure modes and recovery options.

This operational context informed our approach to AI integration, leading us to develop an architecture that could bridge between modern AI capabilities and established operational systems while maintaining the rigorous safety standards required for space operations.

3. State-of-the-art

The integration of AI agents into space operations requires careful consideration of both architectural patterns and orchestration tools. As mission control systems evolve to incorporate AI capabilities, understanding the various agent topologies and available orchestration frameworks becomes crucial for system architects and maintenance teams.

AI agents are software programs utilizing artificial intelligence to perform tasks, make subtle decisions, or tap into their knowledge base to inform users. In our system, these agents can work solo—like a helpful expert—or team up to deliver the best answers together.

Topology is how we organize them: a single agent might handle a task alone, or several could collaborate, pooling their insights. This setup lets our AI tackle tasks, decide smartly, and provide critical feedback to the user.

Agent topologies define fundamental patterns for implementing AI capabilities, each offering distinct trade-offs between simplicity, capability, and reliability. These patterns have emerged from decades of research in autonomous systems and practical implementations across industries. In the context of space operations, where system reliability and predictability are paramount, choosing the appropriate agent topology can significantly impact maintenance workflow effectiveness and system stability.

Complementing these architectural patterns, agent orchestration tools provide the technical foundation for implementing and managing AI agents in production environments. The rapidly evolving landscape of these tools reflects the growing sophistication of AI applications and the increasing need for robust, scalable agent management solutions. Each tool brings its own approach to agent coordination, state management, and system integration, making tool selection a critical architectural decision.

Below we present an analysis of both agent topologies and current orchestration tools, with particular attention to their applicability in space mission control contexts. This analysis informed our own architectural decisions in developing a novel agent management system specifically tailored to space operations maintenance workflows.

AI agents are software programs utilizing artificial intelligence to perform tasks or make decisions autonomously. The concept of "topologies" in this context refers to the structural and interactional arrangements of these agents within a system, particularly how they communicate and are organized.

3.1. Topologies

The design of AI agent systems hinges on their organizational structure, or topology, which dictates how agents interact, share responsibilities, and process information [8]. In the context of space mission control applications, where reliability, scalability, and adaptability are paramount, selecting an appropriate topology is critical to balancing automation with operational safety. Various topologies have been explored in autonomous systems research, each offering unique advantages and trade-offs in terms of complexity, robustness, and integration with legacy infrastructure. Below, we outline five key topologies, ranging from simple single-agent designs to complex swarm-based systems, and assess their applicability to maintenance workflows within mission control environments.

- **Single Agent -** One AI agent handles all tasks without any interaction with other agents.
- **Centralized** - One central AI agent coordinates the actions of multiple subordinate AI agents.
- **Decentralized -** Multiple AI agents interact and make decisions without a central authority.
- **Hierarchical** - AI agents are organized in a hierarchical structure with different levels of authority.

- **Swarm** - Multiple simple AI agents work together through local interactions to achieve complex behaviour.

3.2. Agent Orchestration Tools and Libraries

The practical implementation of AI agents in space mission control requires robust orchestration tools and libraries to manage their deployment, coordination, and integration with existing systems. These tools provide the technical infrastructure for building, scaling, and maintaining agent-based architectures, offering diverse approaches to agent collaboration, state management, and system interoperability. The rapid evolution of these frameworks reflects the growing demand for sophisticated AI applications, yet their applicability to space operations—particularly with legacy systems like SCOS-2000 or SIMULUS—remains a critical consideration. Below, we evaluate several prominent orchestration tools and libraries, assessing their strengths, limitations, and relevance to the maintenance workflows central to our framework. This analysis informed our decision to develop a bespoke solution tailored to the unique constraints of space mission control.

Tools and Libraries Overview

Tool/Library	Description	Pros	Cons
LangChain [4]	A framework for building applications with large language models (LLMs).	Well-maintained; strong NLP capabilities.	Complex abstractions; inconsistent docs.
LangGraph [6]	Extends LangChain with graph-based multi-agent workflows.	Simplifies complex flows; improves error handling.	Adds complexity; steeper learning curve.
CrewAI [9]	Facilitates multi-agent collaboration by assembling agents into teams.	Production-ready; supports teamwork.	Advanced features complex; learning curve.
AutoGen [12]	Optimizes workflows by orchestrating agents for complex tasks.	Efficient automation; flexible communication.	Less structured; limited creative output.
RASA [14]	Open-source framework for conversational AI development.	Customizable; strong community support.	High setup/maintenance overhead.
OpenAI Swarm [15]	Designed for scalable, collaborative multi-agent environments.	Flexible; supports complex interactions.	Resource-intensive; needs careful management.
Microsoft Semantic Kernel [16]	Integrates AI with enterprise-grade features and knowledge management.	Robust integration; enterprise focused.	Complex for small projects; Microsoft-centric.
HuggingFace Transformers Agents [17]	Tools for building transformer-based AI agents.	Extensive library; easy integration.	Resource-heavy; may need fine-tuning.
Botpress [18]	Platform for conversational AI agents with a focus on usability.	User-friendly; multilingual support.	Limited to conversational use; customization heavy.

3.3. Example with Langchain

Langchain provides foundational agent capabilities, such as tool-calling and basic workflows, but is recommended for straightforward chains rather than complex multi-agent systems. The documentation advises using Langgraph for agents, especially for controllable and customizable workflows[19]. For instance, single agent topologies are well-handled by Langchain's AgentExecutor, but for multi-agent scenarios, Langgraph's graph-based approach is superior.

Topology	Description	Implementation in Langgraph	Relation to Langchain
Single Agent	One agent handles all tasks.	A graph with one node, possibly with internal sequences.	Basic agent support via chains and tools.
Centralized	One central agent coordinates multiple subordinates.	Central node connected to multiple nodes, controlling flow.	Limited; better handled by Langgraph for complexity.
Decentralized	Agents interact without central authority, peer-to-peer.	Graph with no central node, e.g., fully connected or ring.	Basic multi-agent support but lacks graph control.

Hierarchical	Agents in a hierarchy, e.g., tree structure.	Graph with tree-like structure, edges representing authority.	Not optimized; Langgraph better for hierarchical flow.
Swarm	Simple agents interact locally for complex behavior, like insect swarms.	Graph with local connections, e.g., grid or network, for updates.	Minimal support; Langgraph excels with local interactions.

3.4. Our unique use-cases

Building on our focus on the maintenance lifecycle (Section 2.2), we identified several key use-cases where AI integration can provide significant benefits. These use-cases [20], demonstrate the practical application of our architecture in addressing real-world maintenance challenges faced by organizations like EUMETSAT and ESA. Implemented and tested using operational satellite simulators (SATSIM) and mission control systems (MCS) for missions such as Sentinel-6, these use-cases ensure that our solutions are grounded in the practical realities of space mission control maintenance. Below, we present an overview of these use-cases and highlight how they leverage the features of our novel agent management framework.

3.4.1. MOFIE (Mission Operations Facility Integration Environment)

This use-case tackles the challenges of managing anomalies and verifying requirements, particularly in relation to warranty issues and identifying duplicate problems. By integrating historical anomaly data and requirements documentation with our agent-based architecture, it can analyse reports to detect patterns, identify potential duplicates, and find applicable requirements.

3.4.2. Automated Anomaly Report Analysis

Here, AI agents automate the processing of incoming anomaly reports, checking for completeness and identifying missing critical information essential for resolution. The function caller agent, a key component of our architecture, interprets natural language inputs from engineers and routes requests to specialized agents handling anomaly data. Additionally, the support for stateful agents ensures that the context of ongoing investigations is preserved, enhancing the accuracy and relevance of the analysis. This use-case exemplifies how our framework streamlines time-intensive anomaly workflows.

3.4.3. Knowledge Base Integration

To address the issue of fragmented documentation across various systems, our architecture integrates multiple knowledge sources into a unified framework accessible via natural language queries. Leveraging Retrieval-Augmented Generation (RAG), AI agents retrieve relevant documentation, procedures, and historical data in response to engineer queries. This capability reduces the time spent searching disparate repositories and ensures access to up-to-date, comprehensive information, directly supported by the architecture's flexible agent composition and knowledge integration features.

3.4.4. Maintenance Procedure Validation

This use-case focuses on automatically validating maintenance procedures to ensure they remain synchronized with system changes—a task traditionally labour-intensive and prone to oversight. Our architecture enables AI agents to cross-reference procedures with current system states and documentation. The environment system facilitates dynamic composition of agents tailored to specific validation tasks, ensuring procedures are checked against the correct mission configurations. This enhances reliability and reduces the validation burden on maintenance teams.

3.4.5. Just Chat

Demonstrating the power of natural language interfaces, this use-case allows engineers to interact with complex systems using plain language—e.g., querying system statuses, retrieving telemetry data, or initiating simulations. The function caller agent interprets these queries and directs them to the appropriate AI or non-AI agents within the environment. This reduces cognitive load, lowers the entry barrier for new team members, and showcases the architecture's ability to bridge human inputs with technical system interactions effectively.

3.5. Reasoning for a framework

The landscape of AI agent frameworks is undergoing unprecedented rapid evolution[11]. LangChain, launched in October 2022, has already seen over 12,000 commits and multiple major architectural changes. During this same period, new frameworks like LangGraph emerged, promising different approaches to agent orchestration. This rapid pace of development, while exciting, creates significant challenges for mission-critical systems that require stable, well-tested foundations.

The volatility in the AI agent ecosystem makes it particularly challenging to select and commit to any existing framework. Today's "best practice" architecture might be obsolete within months as new models, capabilities, and patterns emerge.

For example, the release of GPT-4 and its variants prompted fundamental redesigns in many agent frameworks to accommodate new capabilities. This constant flux makes it especially risky for space operations, where system stability and long-term maintainability are essential.

> *"If your business depended on GPT-3, GPT-4 took its place; if your business revolves around GPT-4, GPT-5 will take over."*

Beyond this inherent instability, current frameworks present additional challenges when applied to space mission control maintenance. These tools assume greenfield development environments where new systems can be built from scratch using modern technologies. However, mission control maintenance must work with legacy systems, some dating back decades, that weren't designed for AI integration.

This creates a high barrier to entry for maintenance teams who are experts in space operations but may not have deep software development experience. While tools like LangGraph and CrewAI offer sophisticated agent collaboration capabilities, they lack the flexibility to easily incorporate existing mission control systems and workflows.

A particularly critical gap exists in resource management. Existing frameworks assume dedicated resources for each agent deployment, which is impractical in space operations where expensive simulators and control systems must be shared across users and teams. Current tools also lack built-in concepts for managing different spacecraft configurations or mission-specific requirements, leading to complex custom implementations.

Knowledge integration presents another significant challenge. While tools like Microsoft's Semantic Kernel provide strong capabilities for knowledge management, they don't address the specific needs of space operations where knowledge is distributed across various specialized systems, procedures, and formats. The inability to easily integrate with existing knowledge bases and maintenance procedures limits the practical utility of current solutions.

These limitations, combined with the rapid evolution of the AI agent ecosystem, pointed to the need for a fundamentally different approach - one that could bridge the gap between modern AI capabilities and the practical realities of space mission control maintenance while remaining adaptable to future technological changes.

4. The Framework

Our research introduces a fundamentally new approach to integrating AI capabilities into mission control system maintenance. Unlike existing agent frameworks that impose rigid structures, require significant coding expertise or, our architecture enables dynamic creation of agent topologies through a simple web interface. This flexibility allows maintenance teams to adapt the system to their specific needs while maintaining operational safety.

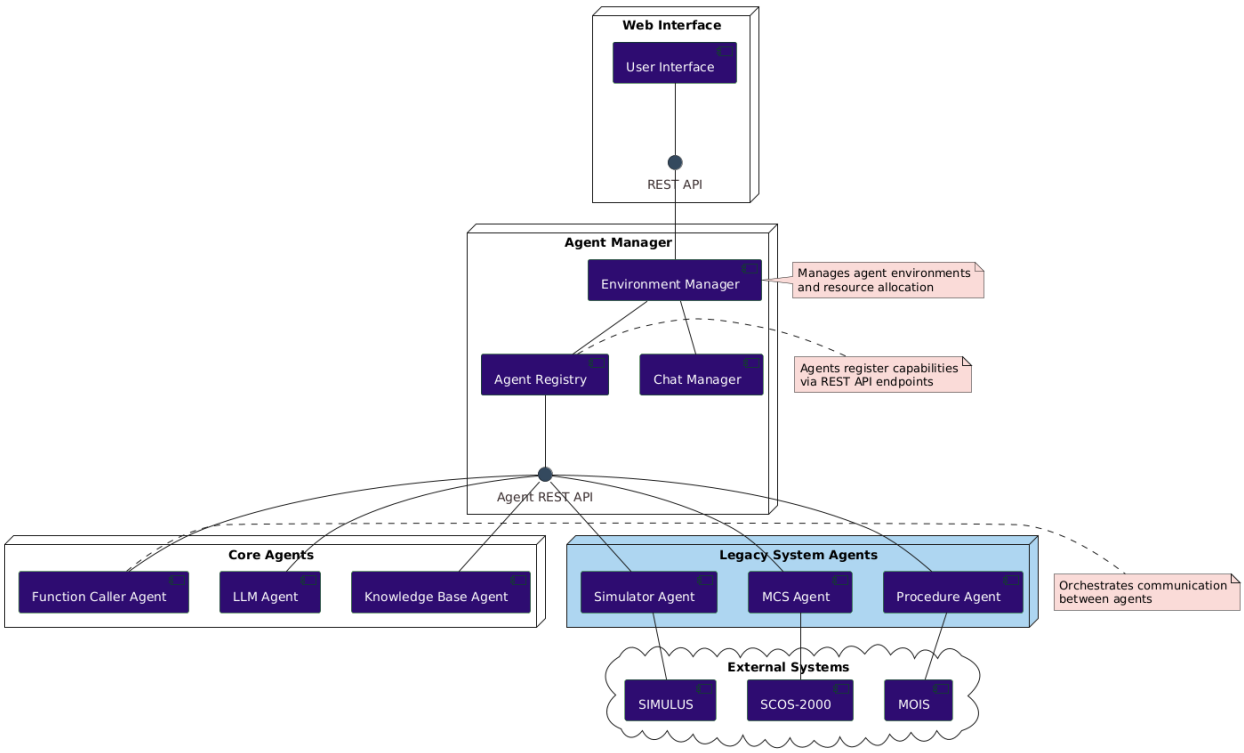

Fig. 2. The Framework's components

The architecture centers around four key innovations:

First, we redefine the concept of an agent as any service exposing a REST API. This simple but powerful abstraction allows seamless integration of both AI and non-AI components. We make extensive use of "*convention over configuration*". By implementing specific endpoint conventions like "fromUser" for chat interaction, existing services can be easily incorporated into the agent ecosystem.

Second, we introduce the concept of meta-agents, which serve as templates for creating specialized agent instances. This allows a single agent implementation to be reused across different missions or spacecraft, significantly reducing development overhead. For example, a simulator meta-agent can be instantiated to work with specific missions like Sentinel-6 or CO2M while maintaining consistent interfaces. This is also practical as mission operators normally share a limited amount of simulators (we cannot raise one new simulator per chat).

Third, our environment system provides unprecedented flexibility in agent composition. Users can create custom environments selecting from available agents through a web interface, without requiring programming knowledge. This allows maintenance teams to quickly assemble the exact set of tools needed for specific tasks.

Fourth, our multi-chat support within environments enables parallel maintenance activities while ensuring resource isolation. Multiple users can interact with the same environment through separate chat interfaces, maximizing resource utilization while preventing conflicts.

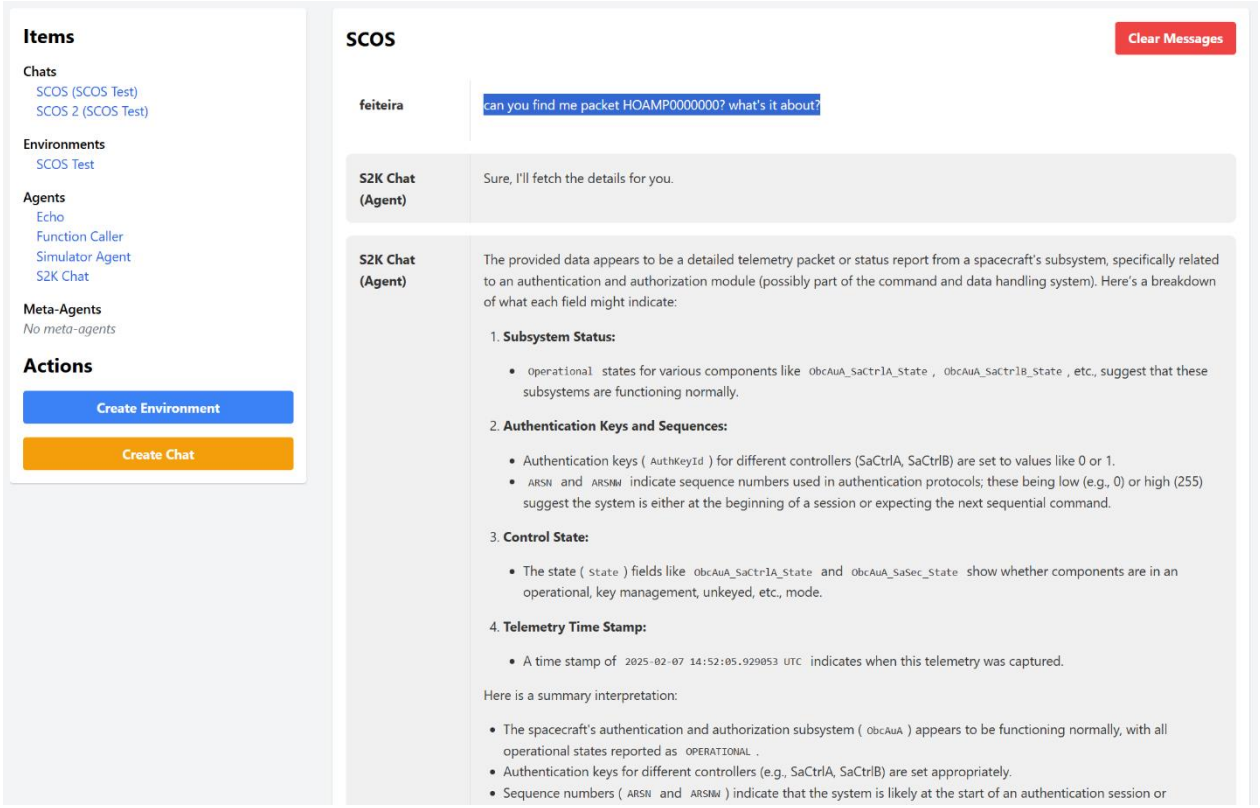

Fig. 3. Screenshot with Spacecraft Control System agent

The implementation follows a layered approach that prioritizes maintainability and extensibility. At its core, the Agent Manager serves as a central registry and orchestrator. It maintains the catalogue of available agents, handles environment creation, and manages chat sessions. New agents can register themselves by exposing standard REST endpoints, with the Agent Manager automatically discovering their capabilities.

A key innovation is our "function caller" agent, which acts as an intelligent bridge between user requests and available services. This agent understands the capabilities of other agents in the environment and can dynamically route requests to appropriate endpoints. This eliminates the need for users to understand the technical details of individual agents while ensuring requests are handled efficiently.

4.1. Approach

4.1.1. Resource Management and Scalability

Our architecture introduces innovative solutions to resource management challenges in mission control system maintenance. Traditional approaches often require dedicated system instances for each maintenance activity, leading to inefficient resource utilization. In contrast, our environment-based model enables multiple maintenance activities to share resources intelligently while maintaining operational isolation.

Resource allocation is handled dynamically through the environment system. When a maintenance engineer creates a new environment, the Agent Manager automatically coordinates resource access across selected agents. This is particularly important for legacy system integration, where limited licenses or hardware constraints might restrict concurrent usage. For example, a single spacecraft simulator instance can be safely shared across multiple maintenance sessions through our agent abstraction layer.

The architecture's REST-based communication model enables efficient scaling as maintenance demands grow. New agent instances can be dynamically provisioned based on workload, while the Agent Manager handles routing and load balancing. This approach has proven particularly effective for AI components, where GPU resources can be allocated based on actual usage patterns rather than peak demand estimates.

4.1.2. Knowledge Integration and Transfer

A significant innovation in our architecture is its approach to maintenance knowledge management. Traditional mission control systems often suffer from knowledge fragmentation across various documentation systems, wikis, and tribal knowledge. Our agent-based architecture addresses this through intelligent knowledge integration and transfer mechanisms.

The system captures maintenance workflows and decisions through chat interactions, automatically building a searchable knowledge base. This is enhanced by our function caller agent, which can correlate current maintenance activities with historical cases, procedures, and documentation. The result is a living knowledge system that grows more valuable over time, helping to preserve institutional knowledge and reduce dependency on individual experts.

Meta-agents play a crucial role in knowledge standardization. By encapsulating common maintenance patterns and procedures, they ensure consistent approaches across different missions while allowing for mission-specific customization. This has proven particularly valuable for anomaly investigation workflows, where standardized approaches improve resolution efficiency.

4.1.3. Practical Applications and Results

Initial deployment of our architecture has demonstrated significant improvements in maintenance efficiency. Anomaly reproduction, traditionally one of the most time-consuming maintenance tasks, has been particularly impacted. The system's ability to automatically coordinate between documentation, simulators, and control systems has reduced reproduction time from days to hours in many cases.

Real-world usage has validated our design decisions around agent flexibility and environment management. Maintenance teams have created specialized environments for different tasks, from routine health checks to complex anomaly investigations. The ability to mix AI and non-AI agents has proven especially valuable, allowing teams to leverage automation where appropriate while maintaining human oversight of critical decisions.

4.1.4. Convention over Configuration

The framework leverages convention-over-configuration principles to enhance flexibility and simplify integration, especially for legacy systems. It defines standardized REST API endpoints, such as `/fromUser` for user interactions and `/updateEnvironmentDetails` for environment updates, enabling seamless incorporation of agents without extensive customization. This approach streamlines onboarding for both AI and non-AI components, allowing even minimally modified services — like a legacy telemetry processing service — to interact with modern AI agents. By reducing development overhead and ensuring consistency across diverse systems, these conventions make the framework well-suited for the heterogeneous environments common in space mission control applications.

*See **Appendix A** Convention-over-Configuration for detailed technical information.*

4.1.5. Stateful and Stateless agents

In our framework, agents are categorized as either stateless or stateful based on whether they maintain internal state between interactions. Stateless agents handle each request independently, without retaining any information from previous interactions. This design allows them to be deployed across multiple environments simultaneously, as their responses are solely based on the input they receive at the moment. A typical example is a document reader agent, which retrieves and presents documents based on user queries without needing to maintain any context or history. This reusability makes stateless agents particularly efficient for tasks that do not require continuity or memory of past actions.

In contrast, stateful agents maintain an internal state that can evolve with each interaction, allowing them to provide context-aware responses or manage ongoing processes. This state might be due to the underlying system (a Simulator agent which bridges one specific simulator is dependent on its state). Due to their reliance on this persistent state, stateful agents must be exclusive to a single environment to ensure that their behaviour remains consistent and predictable. For example, an agent overseeing a spacecraft simulation must track the simulation's current parameters and history, and sharing this agent across environments could lead to conflicting states or erroneous outputs. Our framework supports both stateful and stateless agents, enabling users to select the appropriate type based on the specific requirements of their tasks. This dual support ensures that the system can handle a wide range of applications, from simple, reusable utilities to complex, state-dependent processes, all within a unified architecture.

4.2. Agent Manager

The Agent Manager is the cornerstone of our novel agent management framework, serving as the central hub that orchestrates the interactions between users, agents, and the underlying mission systems. It is designed to ensure seamless coordination, scalability, and adaptability in maintenance workflows while maintaining the operational reliability critical to space mission control applications. By functioning as both a registry of available agents and an active orchestrator of their activities, the Agent Manager enables the dynamic creation and management of agent environments tailored to specific maintenance tasks.

4.2.1. Role and Functionality

At its core, the Agent Manager maintains a comprehensive catalogue of all registered agents within the system, whether AI-driven or traditional non-AI services. This catalogue includes metadata about each agent's capabilities, such as supported endpoints, input/output schemas, and operational roles (e.g., simulation, telemetry processing, or documentation retrieval). The Agent Manager uses this information to facilitate environment creation, where users can select and assemble agents via a web interface to address specific maintenance needs, such as anomaly investigation or procedure validation. It also manages chat sessions, ensuring that user interactions are routed correctly and that responses from multiple agents are coordinated effectively.

4.2.2. Agent Registration and Capability Detection

Agent registration is a streamlined process that leverages the framework's "convention over configuration" principles (see Section 4.1.4). New agents register themselves by exposing standard REST API endpoints, such as `/fromUser` for receiving user messages and `/updateEnvironmentDetails` for receiving environment updates (detailed in Appendix A). Upon registration, the Agent Manager automatically queries these endpoints to discover the agent's capabilities. For example, an agent might expose a specification or a predefined schema that outlines its supported operations—e.g., a simulator agent might indicate it can execute telecommands (TC) or retrieve telemetry (TM) data for specific spacecraft configurations like Sentinel-6 or CO2M.
This approach ensures that even legacy systems, minimally adapted to expose REST interfaces, can integrate into the framework without extensive reconfiguration. For instance, a legacy SCOS-2000 telemetry service might register by exposing a /fromUser endpoint to accept natural language queries, which the Agent Manager then catalogs and makes available for environment composition.

4.2.3. Environment and Session Management

The Agent Manager's orchestration role extends to managing the lifecycle of environments—custom assemblies of agents created for specific maintenance tasks. When a user creates an environment via the web interface, the Agent Manager allocates the selected agents, establishes communication channels, and ensures resource isolation. For stateful agents, such as those tied to a specific simulator instance, it enforces exclusivity to a single environment to prevent state conflicts. For stateless agents, like a documentation retrieval service, it allows reuse across multiple environments, optimizing resource utilization.
Chat session management is another critical function. The Agent Manager maintains separate chat instances within each environment, enabling multiple users to interact concurrently without interference. It routes user inputs to the function caller agent (Section 5.2.6), which interprets the requests and delegates them to appropriate agents based on their registered capabilities. Responses are then aggregated and returned to the user through the chat interface. This process ensures that simultaneous agent responses—a noted limitation (Section 5.3.1)—are minimized by relying on the function caller agent's intelligent routing, though future enhancements may introduce additional coordination mechanisms.

4.2.4. Practical Implementation

In summary, the Agent Manager is a pivotal component that operationalizes the flexibility and adaptability of our architecture. Its automated registration, dynamic orchestration, and scalable design make it an enabler for integrating AI into mission control maintenance, addressing the unique challenges of legacy systems and shared resources in space operations.

4.3. The Function Caller Agent

To enable common use-cases, some core agents are available. The function caller agent is a cornerstone of the system's architecture, acting as an intelligent bridge between users and the diverse array of non-AI agents operating within the framework. Its core responsibility is to streamline user interactions by interpreting natural language inputs—such as queries, commands, or instructions—and efficiently delegating them to the most suitable REST agent for execution. Powered by a large language model (LLMs) [10], the function caller agent parses user requests, discerning their intent, and mapping them to the appropriate functionality offered by the system's agent ecosystem.

For instance, consider a scenario where a technician submits a request like, "tell me if telemetry is flowing into the system" The function caller agent processes this input, identifies that it involves retrieving diagnostic data, and routes it to the agent responsible for handling dynamic configuration variables (MISC dynamic server in the case of SCOS-2000). This translation of user intent into actionable tasks spares users from needing to know the intricate details of individual agent interfaces, APIs, or operational protocols, making the system far more accessible and user-friendly.
In addition to its routing capabilities, the function caller agent is designed to handle complexity and uncertainty in user inputs. When faced with ambiguous requests—say, a command like "check the system status," which could apply to multiple subsystems or agents—the agent can either seek clarification from the user or leverage contextual cues to select the most relevant agent. This proactive resolution of ambiguity helps prevent miscommunication, reduces the likelihood of errors, and ensures that tasks are executed accurately and efficiently.
Through its integration of LLMs, detailed system prompts and intelligent routing, the function caller agent empowers the system to remain flexible, responsive, and aligned with the practical needs of its users, ultimately fostering a more effective and streamlined operational experience.

4.4. Example Workflow

To illustrate the practical application of our framework, this section presents an example workflow for investigating an anomaly in spacecraft telemetry—a critical maintenance task in space mission control. This scenario highlights how the architecture will coordinate AI and non-AI agents to streamline the process, using a natural language interface powered by a Large Language Model (LLM) via the function caller agent. The workflow is designed to reduce the time and effort required for anomaly investigation, a task that traditionally spans hours or days due to manual coordination between documentation, telemetry data, and simulation systems.

4.4.1. Scenario

A maintenance engineer at EUMETSAT needs to investigate an unexpected drop in the value of telemetry parameter X, reported on date Y from a spacecraft such as Sentinel-6. The goal is to identify potential causes, retrieve relevant documentation, analyse telemetry data, and, if necessary, reproduce the anomaly in a simulator.

4.4.2. Workflow Steps

Environment Creation
The engineer accesses the web interface and creates a new environment tailored for anomaly investigation.
Agents Selected:

- Documentation Retrieval Agent (stateless): Searches and retrieves relevant documentation and procedures from the knowledge base (e.g. FARC).
- Telemetry Analysis Agent (stateless): Queries and analyzes telemetry data from the mission database (e.g., SCOS-2000 TM archives).
- Simulator Agent (stateful): Interfaces with a spacecraft simulator (e.g., SIMULUS or SATSIM) to reproduce anomalies, tied to a specific instance for state consistency.
- Function Caller Agent (core agent): Interprets user inputs and routes requests to the appropriate agents using its LLM capabilities.

The Agent Manager allocates these agents to the environment. The simulator agent, being stateful, is exclusively assigned to this environment to maintain its state (e.g., simulation parameters), while stateless agents can be reused across other environments.

Chat Session Initiation

The engineer opens a chat session within the environment and types: "I need to investigate an anomaly where parameter X dropped unexpectedly on date Y."
The web-based chat interface, built with Django Channels, provides real-time interaction.

Interpretation by Function Caller Agent

The function caller agent receives the message and uses its LLM (e.g., Llama 3.1) to interpret the intent: the engineer seeks to investigate an anomaly involving parameter X on date Y.
It determines that the request requires:

- Retrieving documentation related to parameter X.
- Fetching and analysing telemetry data for parameter X around date Y.
- Potentially setting up a simulation if further reproduction is needed.

Delegation to Specific Agents

The function caller agent routes tasks to the selected agents via their REST APIs:

- Documentation Retrieval Agent: Receives a request to search for documents related to parameter X.
- Telemetry Analysis Agent: Receives a request to query telemetry data for parameter X around date Y.

Agent Executions

Documentation Retrieval Agent:

- Searches the knowledge base (e.g., integrated Procedure Management Tool or FARC systems).
- Returns a list of relevant documents, such as parameter specifications or historical anomaly reports.

Telemetry Analysis Agent:

- Queries the telemetry database (e.g., SCOS-2000 archives).
- Analyses the data, identifying the exact time of the drop and any correlated parameters.
- Returns a summary, potentially including graphs of the telemetry trends.

Simulator Agent (if activated later): Remains idle until explicitly invoked.

Response Aggregation

The function caller agent collects responses from the documentation and telemetry agents.
It formulates a comprehensive reply, such as:

- Links to relevant documentation (e.g., PDFs or excerpts).
- A telemetry analysis summary with key findings (e.g., "Parameter X dropped by 15% at 14:32 UTC on date Y, with parameter Z showing a correlated spike").

The response is sent back to the engineer through the chat interface.

Further Interaction

The engineer reviews the information and decides to reproduce the anomaly. They type: "Set up a simulation with conditions similar to date Y."
The function caller agent interprets this request and routes it to the simulator agent.
Simulator Agent:

- Configures the simulation using telemetry data from date Y (e.g., via SIMULUS).
- Starts the simulation and reports its status (e.g., "Simulation running with conditions matching date Y").

Knowledge Capture

All chat interactions, including queries, agent responses, and simulation outcomes, are logged by the Agent Manager.
This creates a searchable knowledge base, capturing the workflow and decisions for future reference (e.g., similar anomalies in Sentinel-6 or other missions).

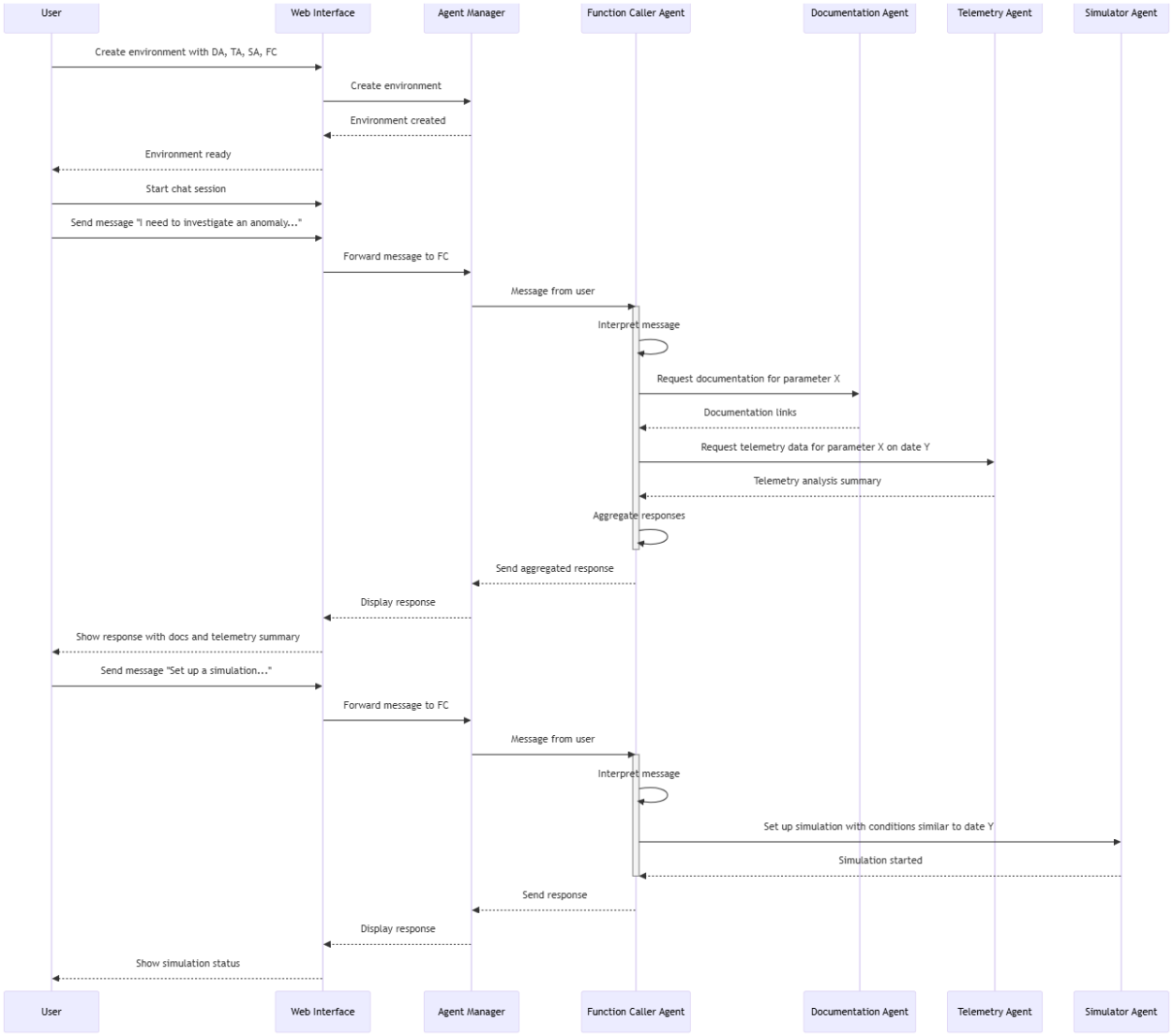

Fig. 4. Flow diagram of example workflow

4.5. Future Directions

The novel agent management framework introduced in this work has shown considerable promise in enhancing maintenance workflows for space mission control applications. However, with the increasing complexity of space missions and rapid advancements in AI technologies, there are numerous opportunities to extend and refine the framework. This section explores key areas for future research and development, aiming to improve coordination mechanisms, scalability, predictive maintenance, and integration with next-generation systems, while addressing the unique demands of space operations.

4.5.1. Advanced Coordination Mechanisms

A current limitation of the framework is the possibility of multiple agents responding simultaneously to the same user input, especially when they share communication channels like the `/fromUser` endpoint. To overcome this, future efforts could focus on developing advanced coordination mechanisms, including:

- Context-Aware Routing: System prompts could be expanded, that that agents evaluate user input context, agent capabilities, and system states, before directing requests to the most suitable agent.

- Priority-Based Systems: Assigning agents priority levels based on their expertise or role—such as giving an anomaly detection agent precedence over a documentation agent—could streamline responses in critical scenarios.
- Decentralized Negotiation Protocols: Agents could negotiate among themselves, sharing confidence levels or proposed actions to determine the best responder, enhancing system reliability without requiring centralized oversight.

These enhancements would ensure smoother, more reliable interactions, vital for the high-stakes environment of space mission control.

4.5.2. Scalability and Orchestration

As space missions grow in scope, the framework must scale to manage an increasing number of agents effectively. Future research could explore:

- Hierarchical Agent Structures: Organizing agents and/or environments into layers—lower-level agents for routine tasks and higher-level agents for coordination—could simplify interactions and boost scalability.
- Self-Organizing Agents: Enabling agents to adapt their behaviour autonomously could create resilient systems capable of managing unexpected challenges without human intervention.

These improvements would allow the framework to support larger missions while maintaining responsiveness and dependability.

4.5.3. Predictive Maintenance and Proactive Monitoring

The framework's current strengths can be expanded to include predictive maintenance and proactive monitoring, leveraging AI to anticipate issues before they arise. Potential developments include:

- Automated Procedure Generation and Validation: AI agents could generate and simulate maintenance procedures in virtual environments, identifying issues before implementation in live systems.
- Intelligent Resource Allocation: Agents could optimize resources for maintenance tasks based on predicted needs and mission priorities, improving operational efficiency.

This shift from reactive to proactive capabilities would reduce downtime and mitigate risks of mission-critical failures.

4.5.4. Integration with Next-Generation Mission Control Systems

To remain compatible with evolving mission control architectures like the European Ground Systems Common Core (EGS-CC), the framework must adapt. Future directions include:

- Support for Distributed Simulation: Enhancing compatibility with tools like SIMULUS would allow complex, mission-specific simulations for procedure validation and anomaly analysis.
- AI-Driven Automation in Maintenance Workflows: Extending automation to real-time tasks—such as telemetry analysis and decision support—could broaden the framework's utility beyond maintenance.

These adaptations would ensure the framework's relevance as space operations transition to advanced systems.

4.5.5. Knowledge Management and Transfer

The framework's ability to capture maintenance workflows offers a foundation for robust knowledge management. Future enhancements could include:

- Automated Knowledge Base Curation: AI agents could extract and update insights from maintenance activities and anomaly reports, keeping the knowledge base current and reducing manual effort.
- Cross-Mission Knowledge Transfer: Standardized workflows and meta-agents could share lessons across missions, improving efficiency organization-wide.

- Natural Language Interfaces for Knowledge Retrieval: Advanced natural language processing could enhance knowledge access, enabling engineers to quickly retrieve critical information.

These developments would preserve institutional knowledge and reduce reliance on individual expertise.

4.5.6. Addressing Ethical and Safety Considerations

As AI assumes greater roles in mission-critical systems, ethical and safety issues must be addressed. Future work should focus on:

- Explainability and Transparency: Making AI decisions transparent and understandable would build trust and allow human verification of critical actions.
- Fail-Safe Mechanisms: Robust fallback options, such as reverting to human oversight during uncertainty, would maintain safety in case of AI failures.

These measures would safeguard the reliability and integrity of space operations as AI integration deepens.

4.6. Limitations

The current framework, while innovative, still faces limitations that could impact its functionality, scalability, and user experience in space mission control applications. These limitations highlight areas for improvement and future research to ensure the system's reliability and adaptability in critical operational environments.

4.6.1. Simultaneous Agent Responses

One significant limitation of the current framework is the potential for multiple agents to respond simultaneously to the same user input, particularly when multiple agents are subscribed to the same communication channel, such as the `/fromUser` endpoint. This can lead to overlapping or conflicting responses, which may confuse the user or result in inconsistent system states. For instance, if two agents interpret a user's command differently and take conflicting actions, it could disrupt the maintenance workflow or even compromise system integrity in critical scenarios.

To mitigate this issue, the framework could benefit from a more sophisticated coordination mechanism. While the function caller agent provides some level of request routing, it may not be sufficient in all cases, especially when user inputs are ambiguous or when multiple agents have overlapping capabilities. Implementing an interceptor or buffer agent that acts as a frontend to the user could help manage the flow of responses, ensuring that only the most appropriate agent responds or that responses are presented in a coordinated manner.

4.6.2. Scalability Challenges

Related to the point above, another limitation relates to the scalability of the system. As the number of agents in an environment increases, the complexity of managing their interactions grows exponentially. Without proper coordination mechanisms, the risk of conflicts and inefficiencies rises, potentially degrading system performance and reliability. Future iterations of the framework will need to incorporate some kind of orchestration techniques, such as hierarchical agent structures or decentralized negotiation protocols, to ensure that the system remains manageable and efficient even with a large number of agents.

4.6.3. Integration of AI and Non-AI Agents

The integration of both AI and non-AI agents presents unique challenges due to their differing operational paradigms. AI agents, particularly those leveraging large language models (LLMs), can adapt to new situations and handle ambiguous inputs more gracefully. In contrast, non-AI agents may rely on predefined rules and lack the flexibility to handle unexpected scenarios. Right now, these two types of agents can work together, however observation is required as users create new (unexpected) environments that may create collisions. In the future, the framework may need to provide additional standardization to bridge the gap between AI and non-AI components, to ensure interactions are consistent and reliable.

4.6.4. Architectural Trade-offs

The choice between using a function caller agent versus wrapping all REST APIs in LLM agents represents a fundamental architectural decision with significant implications. While the function caller approach offers a centralized way to manage requests, it may become a bottleneck or a single point of failure, especially under high workloads. On the other hand, wrapping each API in an LLM agent could distribute intelligence across the system but might introduce unnecessary complexity and computational overhead, particularly for simple services. For example, wrapping a straightforward telemetry retrieval API in an LLM agent will increase resource demands without necessarily providing significant benefits. Finding the right balance between these approaches is crucial for optimizing system performance and maintainability.

*4.6.5. Self-*Organization *and Conventions*

The framework also raises questions about whether agents should self-organize to resolve conflicts or ambiguities autonomously. Self-organization could reduce the need for centralized coordination, but it might add complexity and unpredictability. Similarly, the reliance on convention-over-configuration principles (e.g., standard endpoints like `/fromUser`) simplifies integration but may limit flexibility for agents with unique requirements. Exploring whether these conventions should be extended or relaxed, and how they impact system behaviour, is an important area for future investigation.

5. Discussion

AI's growth in the space domain is incoming and inevitable [13]. The novel agent management framework introduced in this work represents a significant step forward in integrating artificial intelligence (AI) capabilities into space mission control applications, particularly within the maintenance lifecycle. By addressing the unique challenges posed by legacy systems, and complex maintenance workflows, the framework provides a pragmatic and flexible solution for organizations seeking to enhance their maintenance processes without disrupting critical operations. This discussion explores the significance of the framework's contributions, its practical implications, and its potential for future expansion, while relating it to existing literature and highlighting areas for further research.

5.1. Bridging Legacy Systems and Modern AI

One of the key contributions of this work is the redefinition of agents as services exposing REST APIs, which facilitates the seamless incorporation of both AI and non-AI components. This approach is particularly valuable in the context of space operational software, where legacy systems—such as SCOS-2000 and SIMULUS—have evolved over decades and often rely on specialized protocols and interfaces [6].
By adhering to a "*convention over configuration*" principle, the framework minimizes the need for extensive modifications to existing services, thereby reducing the risk and cost associated with integration. For example, a telemetry processing service implemented decades ago can be integrated into the agent ecosystem simply by exposing standardized REST endpoints, without requiring significant refactoring.

This approach contrasts with existing AI agent frameworks like Langchain and CrewAI, which often assume greenfield development environments and require some level of coding expertise. In mission control applications, where engineers are experts in using the operational software (and experts in space operations) but may not have deep software development experience, this simplicity is crucial. The framework's web-based interface for environment creation and agent composition further empowers maintenance teams to adapt the system to their specific needs without requiring programming knowledge, lowering the barrier to adoption and enabling incremental integration of AI capabilities.

5.2. Flexibility Through Meta-Agents and Environment-Based Resource Management

The introduction of meta-agents enhances the framework's flexibility and reusability. By serving as templates for creating specialized agent instances, meta-agents enable a single implementation to be adapted for multiple missions or spacecraft configurations. For instance, a simulator meta-agent can be instantiated to work with specific missions like Sentinel-6 or Meteosat Third Generation (MTG) while maintaining consistent interfaces. This not only reduces

development overhead but also promotes standardization across different operational contexts, which is essential for maintaining consistency and reliability.

Complementing this, the environment-based resource management system addresses a critical challenge in mission control maintenance: the efficient utilization of shared resources. Traditional approaches often require dedicated system instances for each maintenance activity, leading to suboptimal resource allocation. In contrast, our framework allows multiple users to share resources—such as simulators and control systems—through isolated environments, enabling parallel maintenance workflows while ensuring operational isolation. This capability is particularly important given the limited availability and high cost of resources in space operations. For example, a single satellite simulator (SATSIM) instance can be safely shared across multiple maintenance sessions, maximizing resource utilization and reducing operational bottlenecks.

5.3. Enhancing Usability with the Function Caller Agent

Central to the framework's usability is the function caller agent, which acts as an intelligent bridge between user requests and available services. By understanding the capabilities of other agents in the environment and dynamically routing requests, the function caller simplifies user interactions and abstracts away the underlying complexity of the system. This is especially beneficial for maintenance engineers who may not possess deep expertise in AI or software development, as it allows them to focus on their core tasks—such as anomaly investigation and procedure validation—without needing to navigate intricate technical details.

The function caller agent's role also mitigates some of the limitations identified in section 4.6, such as the potential for multiple agents to respond simultaneously to the same input. By acting as a dispatcher, it can route ambiguous requests to the most appropriate agent or prompt the user for clarification, reducing the risk of conflicts. However, as discussed earlier, further enhancements—such as priority-based routing or context-aware decision-making—are needed to fully address this challenge.

5.4. Practical Implications and Results

Initial deployment of the framework has demonstrated its potential to improve maintenance efficiency, particularly in the area of anomaly reproduction. Traditionally, anomaly investigation is one of the most time-consuming maintenance tasks, often requiring engineers to coordinate between documentation, simulators, and control systems. By automating these interactions, the framework has reduced reproduction time from days to hours in many cases. For example, an engineer investigating a telemetry anomaly can use natural language commands to instruct AI agents to retrieve relevant documentation, configure the simulator, and execute test scenarios, streamlining the process and enhancing operational efficiency.

These results validate the framework's design decisions around agent flexibility, environment management, and knowledge integration. Maintenance teams can create specialized environments for various tasks, from routine health checks to complex anomaly investigations. The ability to mix AI and non-AI agents has proven especially valuable, allowing teams to leverage automation where appropriate while maintaining human oversight of critical decisions. This balance is crucial in space operations, where safety and reliability are paramount.

5.5. Comparison with Existing Frameworks

When compared to existing AI agent frameworks, our approach offers distinct advantages for the space mission control domain. Langgraph, for instance, provides graph-based workflows for multi-agent systems, but it lacks the flexibility to easily incorporate legacy systems and requires significant coding expertise. Similarly, CrewAI supports multi-agent collaboration but assumes dedicated resources for each agent deployment, which is impractical given the resource constraints in space operations. In contrast, our framework's environment-based approach enables dynamic agent composition and intelligent resource sharing, making it more suitable for maintenance teams who need to quickly adapt to different tasks without disrupting operations.

Additionally, the framework's ability to integrate distributed knowledge bases—such as documentation systems, wikis, and procedure repositories—sets it apart from tools like Microsoft's Semantic Kernel, which focus on general-purpose

knowledge management but do not address the specific needs of space operations. By capturing maintenance workflows and decisions through chat interactions, the system builds a living knowledge base that grows more valuable over time, helping to preserve institutional knowledge and reduce dependency on individual experts.

5.6. Addressing Limitations and Future Directions

As discussed in section 4.6, the framework is not without its limitations. The potential for multiple agents to respond simultaneously to the same input poses a risk of conflicts or confusion, particularly in scenarios with ambiguous user requests. While the function caller agent mitigates this to some extent, further research is needed to develop more robust coordination mechanisms. For example, with increasing context lengths and improved language models, higher complexity system prompts can be generated that better condition the agents.

Scalability challenges also warrant further investigation. As the number of agents in an environment increases, advanced orchestration techniques—such as hierarchical agent structures or decentralized negotiation protocols—may be necessary to maintain system performance and reliability. Exploring these approaches could enhance the framework's ability to handle large-scale deployments, ensuring that it remains effective as space missions grow in complexity.

5.7. Hallucinations

The integration of Large Language Models (LLMs) into our agent-based architecture introduces the phenomenon of hallucinations—instances where the AI generates plausible but inaccurate or fabricated outputs. Understanding and managing hallucinations is critical in the context of space mission control maintenance, where precision and reliability are non-negotiable. Our observations indicate that the occurrence and impact of hallucinations depend heavily on the nature of the tasks assigned to the AI agents, with distinct patterns emerging that inform both our current implementation and future refinements.

In our framework, hallucinations are minimal when tasks are well-defined and unambiguous—such as retrieving a specific telemetry parameter or validating a documented procedure against a known system state. For instance, when a user requests, “What is the current value of TM parameter X?” the function caller agent (Section 4.3) routes the query to the appropriate telemetry agent, and the response is grounded in verifiable system data. In these cases, the task’s clarity and the structured nature of the underlying data prevent the LLM from deviating into speculative outputs. This suggests that hallucinations are not an inherent flaw in the system but rather a function of input complexity and task openness.

The most significant challenges arise with open-ended or poorly constrained queries, such as “What might be causing this anomaly?” Here, the LLM may attempt to synthesize answers based on incomplete context or extrapolate beyond the available data, leading to potential inaccuracies. In some instances, we observed hallucinations manifesting as references to non-existent REST API endpoints. For example, the function caller agent might interpret a vague user request like “adjust the system” as a command to invoke a non-existent endpoint (e.g., `/adjustSystemState`), resulting in a system failure when the request cannot be fulfilled. These failures, however, stem from the agent’s attempt to align with the user’s intent rather than a misunderstanding of the system’s operational boundaries.

6. Conclusions

Our agent-based architecture represents a significant advance in the mission control system maintenance toolkit. By combining flexible agent composition, intelligent resource management, and comprehensive knowledge integration, we've created a system that enhances maintenance efficiency without changing legacy software. The architecture's success in practical deployment demonstrates the value of our approach to balancing innovation with reliability in critical space systems.

The system's ability to dynamically adapt to maintenance sets it apart from existing solutions. As the complexity of space missions continues to grow, the ability to adapt and scale maintenance processes will become increasingly critical. This framework offers a solid foundation for meeting those challenges, reducing technical barriers and empowering teams to reimagine operations enhanced by AI capabilities. By providing a pragmatic path for incremental adoption, it enables organizations to leverage AI technologies without requiring complete system overhauls, paving the way for more efficient, reliable, and innovative space mission control applications.

The future directions outlined here illustrate the framework’s potential to evolve alongside the demands of space mission control. By advancing coordination, scalability, predictive maintenance, and integration with modern systems,

the framework can remain a versatile, reliable tool for maintenance teams while – potentially - expanding to broader operational roles. As AI and space missions grow more intricate, these enhancements will be essential to ensuring efficient, safe, and innovative space operations.

Appendix A Convention-over-Configuration

RESP API definition for an agent that receives messages directly from the users.

```
openapi: 3.0.0
info:
  title: Simple Chat Agent API
  version: 1.0.0
paths:
  /fromUser:
    post:
      summary: Process a message with LLM
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                chat_id:
                  type: integer
                source:
                  type: string
                message:
                  type: string
                history:
                  type: array
      responses:
        '200':
          description: Message processed
```

RESP API definition for an agent that receives environment details (e.g. the Function Caller agent)

```
  /updateEnvironmentDetails:
    post:
      summary: Update environment details
      description: Receive updates about the environment and other agents
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                agents:
                  type: array
                  items:
                    $ref: '#/components/schemas/Agent'
                environment:
                  $ref: '#/components/schemas/Environment'
      responses:
        '200':
          description: Successful update
          content:
            application/json:
```

```
              schema:
                type: object
                properties:
                  status:
                    type: string
                    enum: [success, error]
                  message:
                    type: string

components:
  schemas:
    Agent:
      type: object
      properties:
        id:
          type: integer
        name:
          type: string
        description:
          type: string
        endpoint:
          type: string
          format: uri
    Environment:
      type: object
      properties:
        id:
          type: integer
        name:
          type: string
        system_prompt:
          type: string
        created_by:
          type: integer
          description: User ID of the creator
        created_at:
          type: string
          format: date-time
```

References

[1] Using Artificial Intelligence for Space Challenges: A Survey, March 2025, https://www.mdpi.com/2076-3417/12/10/5106

[2] Survey of machine learning techniques in spacecraft control design, https://www.sciencedirect.com/science/article/pii/S0094576521002514

[3] ESA Operations software products, March 2025, https://essr.esa.int/project/esa-operations-software-products

[4] LangChain, "LangChain Framework Documentation," https://www.langchain.com/

[5] European Ground System - Common Core (EGS-CC), March 2025, https://en.wikipedia.org/wiki/EGS-CC

[6] LangGraph, "LangGraph Documentation," https://langchain-ai.github.io/langgraph/

[7] ESA/ESOC Operational Software Systems, March 2025, https://esoc.esa.int/services-software

[8] Multi-agent system topologies, March 2025, https://en.wikipedia.org/wiki/Multi-agent_system

[9] CrewAI, "CrewAI Documentation," https://crewai.com/

[10] An Effective Query System Using LLMs and LangChain, March 2025, https://www.researchgate.net/publication/372529063_An_Effective_Query_System_Using_LLMs_and_LangChain

[11] Using Artificial Intelligence for Space Challenges: A Survey, https://www.mdpi.com/2076-3417/12/10/5106

[12] AutoGen, "AutoGen Framework," https://microsoft.github.io/autogen/

[13] Space missions out of this world with AI, 23 March 2023, Nature Machine Intelligence, https://www.nature.com/articles/s42256-023-00643-3

[14] RASA, "RASA Open Source," https://rasa.com/docs/rasa/

[15] OpenAI, "Swarm Framework," https://github.com/openai/swarm (Note: As an experimental framework, Swarm's primary documentation is on GitHub; no standalone website exists yet.)

[16] Microsoft, "Semantic Kernel," https://github.com/microsoft/semantic-kernel (Official documentation hosted via GitHub; Microsoft's main site redirects here.)

[17] Hugging Face, "Transformers Agents," https://huggingface.co/docs/transformers/agents

[18] Botpress, "Botpress Platform," https://botpress.com/

[19] Build an Agent | 🦜🔗 LangChain, March 2025, https://python.langchain.com/docs/tutorials/agents/

[20] Maintenance lifecycle of mission control applications: can AI help?, D. Vicente et al, SpaceOps 2025