Article

Not peer-reviewed version

HyMOVE: Versatile Core Avionics Ranging from Satellites to Launchers

Sergio Montenegro * and Moritz Arbab *

Posted Date: 13 January 2025

doi: 10.20944/preprints202501.0857.v1

Keywords: avionics; launcher; satellite; task-migration; framework

Article

HyMOVE: Versatile Core Avionics Ranging from Satellites to Launchers

Sergio Montenegro [1,*] and Moritz Arbab [1]

[1] Department of Aerospace Computer Science, University of Würzburg

* sergio.montenegro@uni-wuerzburg.de

Abstract: Within the HyMOVE project, we are developing a modular orbital transfer vehicle adaptable to various missions. A key innovation is the development of a versatile core avionics system, configurable for satellites, payloads, orbital transfer vehicles, and launchers. Designed for high reliability in launchers and high availability in satellites, this system can adapt to different mission requirements and recover from hardware failures or errors. Unlike fixed designs, the system is highly configurable to meet varying dependability, size, power, and computational needs.

Keywords: avionics; launcher; satellite; task-migration; framework

1. Introduction

Within HyMOVE HyImpulse Technologies GmbH and the Julius-Maximilians-University of Würzburg are developing an orbital transfer vehicle that can be adapted to a wide range of different missions thanks to its modular architecture [1,2]. This modularity will allow a possible standardisation of parts and thus a reduction of costs, by allowing a more complete transfer of processes from general industrial practices, such as serial production, to the space industry.

We do not know of any other core avionics system that can be used for satellites - platform (bus), satellite payload, orbital transfer vehicles and launchers. Our task in the Hymmove project was to develop such a core avionics system.

The HyMOVE avionics is a versatile core avionics system that can be configured (tailored) for different missions and, due to the high reliability requirements of launchers and satellites, has a very high degree of adaptability to reconfigure itself to different situations, e.g. after hardware failures or detected errors. Hymove must provide high reliability for launchers and high availability for satellites. For the launchers, no visible failure or service interruption is allowed from ignition to orbit injection. This is a very short period of between 10 and 30 minutes, but during this time the highest possible readability is mandatory. Satellite operations, on the other hand, require only high availability, but for a period of between one and 15 years. During this time, short interruptions in operations can be tolerated, but the core avionics system must guarantee recovery and, in particular for payload control, data integrity.

We are not aiming for a fixed avionics structure that can be installed in a launcher and/or a small satellite. We want a configurable system that can be tailored to different dependability requirements, for different sizes, computing power and power consumption.

2. From Launcher to Satellite, Reliability to Availability

One of the core technologies of the HyImpulse vehicle is the use of hybrid rocket propulsion: it uses both solid and liquid components. This offers several advantages over both liquid and solid propulsion, the most important being safety and simplicity. Unlike solid rocket engines, hybrid rocket engines are controllable, allowing more complex missions to be carried out using this type of engine [3].

 doi:10.20944/preprints202501.0857.v1

The HyImpulse launcher features an Orbital Transfer Vehicle (OTV) designed to carry two types of payload [4]: deployable and non-deployable. Deployable payloads, typically satellites, can be placed in different orbits as the OTV manoeuvres to reach and deploy them. This process takes less than a day and requires high reliability of the avionics. Once all the "deployable payloads" have been separated, the OTV can remain in orbit and operate as a normal satellite for several years. During this period, the second set of "non-deployable payloads" will be operated. These payloads are "normal" satellite payloads and the OTV has now become a "normal" satellite. During this period, high availability of the avionics is required. At the end of the mission, which may last decades, the OTV can be deorbited using its own propulsion system.

3. Avionics Versatility

To meet this wide range of different and sometimes conflicting requirements for the core avionics of the satellite bus, satellite payload, orbital transfer vehicles and launchers, we had to create:

1. Dynamic adaptability for fault handling, reliability and speed.
2. Static tailoring for different missions.
3. Communication protocols with location transparent partners.

We have developed HyMOVE as a distributed modular system, a (real time capable) network of software and hardware components, where software applications are not fixed on a single piece of hardware, but can flow dynamically to where they can be executed. The software is based on RODOS (Real Time On board Dependable Operating System) [5] and its integrated publish-subscriber [6] middleware. This creates a dependable real-time interconnection network (software and hardware) connecting many unreliable redundant components such as sensors, actuators, communication devices, computers and storage elements. Migration of services is transparent to applications and is handled autonomously by RODOS and its middleware. It is transparent whether the communicating applications are (currently) on the same computer, on different computers in different sub-networks. Component failures are detected, the affected device is disabled and its function is taken over by a redundant component. Components can be switched on and off at any time, and the whole system adapts autonomously to its new configuration in order to continue to fulfill its tasks.

4. Dependability

Any small failure can bring down the whole system. And we can be sure that they will come! We must be ready to detect them and find an alternative configuration to continue working. It is not effective to try to avoid internal failures, despite all efforts they will come. What is effective is to detect and recover in a very short time. We aim to detect and recover from any internal failure in less than 0.5 seconds. We aim for ultra-fast recovery. Any computer on our network can reboot and recover in less than 300ms. If we are able to redistribute tasks between computers, then we can tolerate computer failures at a rate of two times per second without service interruption.

To summarise, dependability is the ability of a system to provide a reliable, consistent and trustworthy service under both normal and adverse conditions [7]. The system always does the right thing at the right time. Sounds simple! Let's make it simple! Just one more world to go. 100% dependability is not possible. There will always be a residual risk, we just have to reduce it to a tolerable level (see Figure 1).

 doi:10.20944/preprints202501.0857.v1

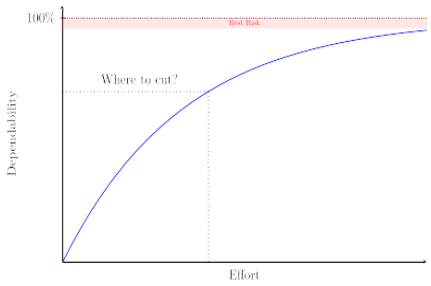

Figure 1. Dependability vs. Effort.

5. Implementation Targets

We are targeting the following development objectives to achieve the proposed dependability and versatility:

1. Dynamic reconfiguration (adaptability) for reliability: Hymove must be able to reconfigure its hardware and software autonomously to adapt to different situations and computational retirements. For example, to increase or decrease hot redundancy to handle different safety-critical phases, to increase or decrease computing power for complex payload calculations and, most importantly, to handle failures by isolating failed units and transferring current functionality to available units. Each hardware unit can be used as cold redundancy, hot redundancy [8,9] , spare unit and high performance reserve. Software tasks can be migrated from one machine to another at any time. Depending on the hardware available, hot redundancy can be implemented in different computers (space redundancy) or as replicated tasks in the same computer (time redundancy) [10]. The same hardware resources can be used for speed or resilience.
2. Static configuration (tailoring) for different missions: Hymove can be used to control launchers, Orbit Transfer Vehicles (OTVs), as a bus (platform) controller for satellites, and as a payload computer for satellites and OTVs. It is interesting to note that it can be both bus and payload computer at the same time. The decision as to what functions and capabilities a specific Hymove system should have is made at design time. At an early stage of development, Hymove can be considered as a set of ready-to-use software and hardware building blocks and a real-time communication interconnect system (network) of different sizes.
3. Connectionless and location independent communication protocol: To be able to replicate and migrate tasks instantly, we need communication protocols where the location and number of communicating agents is transparent. We have developed a Real Time Publish Subscriber middleware (RODOS Real Time Operating System and Middleware), where it does not matter if the communication partners are in the same computer, in different computers in the same subnet, in different subsets and even inside and outside the vehicle, for example for communication between applications running in the vehicle and the ground station. This allows us to distribute tasks between space and ground, simplifying command and telemetry tasks in an incredible way.
4. Concurrent and simultaneous monitoring: In the case of hot redundancy, the computers monitor each other and in the event of a failure, a reset/recovery is forced immediately. In cold redundancy, one computer (worker) keeps its partner (spare) off as long as it (the worker) is working properly. If it detects a failure or crashes, the partner computer is automatically switched on and the newly started computer switches off the failed one and takes over.
5. Irreducible complexity: because complexity is the source of many failures and accidents, our first directive is to reduce complexity to a minimum, as Einstein once said: Make everything as simple

 doi:10.20944/preprints202501.0857.v1

as possible, but not simpler. We rethink every component, especially software components, many times until we have the impression that it cannot be simpler. The basic/infrastructure software, the RODOS operating system and its middleware, has been reimplemented 3 times. Each time simpler than the one before and this process continues. The current implementation of the real-time kernel and middleware is under 4K lines of code.

6. Ultra fast recovery: Since we know that every computer will crash (some day/some second), we must be ready to migrate tasks and recover the crashed computer in the shortest possible time to bring it back into the pool of available resources. Thanks to our philosophy of irreducible complexity, rebooting and re-initialising a computer is very fast. We are able to recover a computer and bring it back into the network in less than 300ms. Critical functions are replicated multiple times across the network in different nodes so that a crash does not jeopardise the flight. The crashed computer is then ready to resume operations in a second.

6. Implementation

6.1. Software: Building Blocks Execution Platform

The main risk factors in a typical core avionics development are the complexity of software interfaces and the difficulty of handling many different interfaces in a single system. The HyMOVE avionics concept addresses these issues and aims to provide a very simple integrated solution of software and hardware.

Our strategy for implementing complex software (control) systems is to break the system down into simple communicating building blocks (BB). The software BBs are similar to integrated circuits: only the interfaces (pin-out) and behaviour need to be known in order to connect them. They can be thought of as ports (like hardware pins) on which the BB expects or publishes services in the form of messages. In our approach, the core avionics system becomes a distributed computer system. No single node is required to be reliable. The nodes are connected by a dependable Real Time Ethernet network, which is the heart of the system. Software services can be distributed across all the computer nodes and can migrate from one node to another, for example in the event of node failure, overload or for power management purposes. In this way, it is possible to assemble a reliable system from unreliable parts. The software network is based on a publisher/subscriber protocol implemented in RODOS as software middleware.

6.2. Rodos and Its Middleware

RODOS (Real Time On board Dependable Operating System) [5] is an open source building block execution platform/environment designed for space and high reliability applications. Simplicity is our main strategy for achieving dependability, as complexity is the cause of most development failures. The system has been developed in C++, using an object-oriented framework that is simple enough to be understood and applied in different application domains. Although the aim is to minimise complexity, no basic functionality is missing, as its microkernel provides support for resource management, thread synchronisation and communication, input/output and interrupt management. The system is fully preemptive, using priority-based scheduling and round-robin for threads with the same priority. On top of this kernel, the RODOS middleware distributes messages locally and globally using gateways.

RODOS provides a (software) interconnection network between applications / building blocks (the middleware). A module needs some services (incoming messages) to be able to provide other services (outgoing messages). The execution platform distributes these services (messages) from producers to consumers. (see Figures 2 and 3).

 doi:10.20944/preprints202501.0857.v1

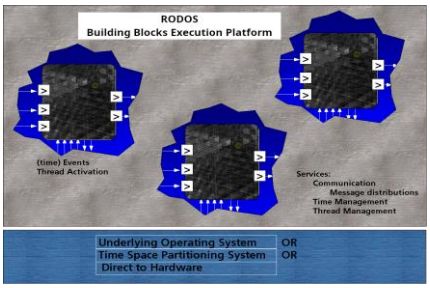

Figure 2. RODOS as building blocks execution platform.

RODOS can run on top of other operating systems or TSPs (Time Space Partitioning Systems) [11], or directly on the hardware if no other operating system is running on the target hardware. In all cases, the interfaces to the devices (or applications) remain the same and a network of applications can run on different platforms and operating systems without modification.

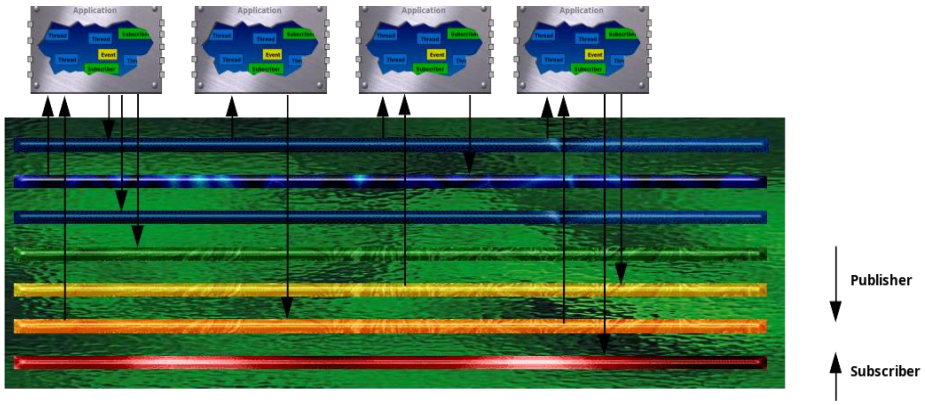

Figure 3. Example of applications communicating using topics.

RODOS provides middleware that enables transparent communication between applications and computing nodes. All communication in the system is based on the Publisher/Subscriber protocol. Publishers publish messages under a given topic. Subscribers (zero, one or more) to a given topic receive all messages published under that topic. Examples of topics could be: position, temperature, and attitude. A topic is represented by a topic ID and a data type. On the software side, the middleware implements an array of topics, which can be compared to hardware buses. The middleware distributes messages to all participants in the system, regardless of their position. For the publisher, it is transparent whether the communication partner is running in the same computer or in another computer connected to the network. We can even build distributed systems where one part runs on the spacecraft and the other on the ground segment. Even these physical barriers will be transparent to the publisher-subscriber communication. So we can dynamically distribute applications between computers. This simplifies fault tolerance and redundancy management. Applications will then be the building blocks (like chips on hardware) and they will be connected by the middleware (like on a circuit board). To build complex software, we "just" plug different BBs together and do not care about their internal structure. To go beyond the hardware boundaries of computing nodes, we use gateways that can read all topics and forward them to the network and vice versa (Figure 4).

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 13 January 2025 doi:10.20944/preprints202501.0857.v1

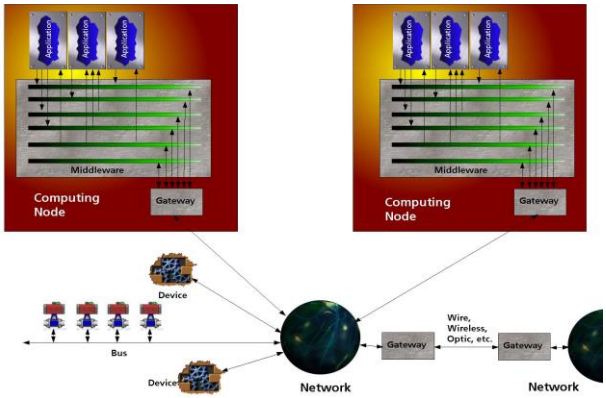

Figure 4. This is a figure. Schemes follow the same formatting.

RODOS provides the means to control timing through thread execution, and the same timing control can be used to publish and distribute messages. Using this feature we can create a software controlled time-triggered communication system. In the case of Hymove, we have an Ethernet network between computing nodes that is used as a software-controlled, time-triggered Ethernet network. The system employs IEEE 802.1AS (gPTP, 2020) for precise time synchronization across distributed nodes. This protocol ensures sub-microsecond accuracy, a critical feature for maintaining consistency and reliability in real-time operations across the avionics network [12]. With this approach, no fixed communication paths are established and the system can be easily reconfigured at runtime. For example, multiple replicas of the same software can run on different nodes and publish the result using the same topic without knowing each other. A voter can subscribe to this topic and vote for the correct result. The core of the middleware only distributes messages locally, but using the integrated gateways to the NetworkCentric" network, messages can reach any node and any application on the network. Figure 5 shows a typical satellite configuration of applications and topics.

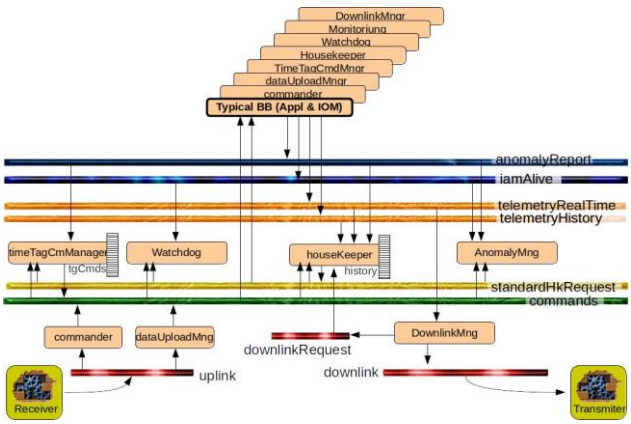

Figure 5. Typical satellite application.

A key feature of Hymove is task migration. The idea is to detect a node failure and then automatically migrate all running tasks from the failed node to a healthy node. For critical tasks, this migration process may take too long (up to several hundred milliseconds depending on the hardware). In this case, the publisher/subscriber method of the middleware offers a great advantage: there can be several instances (replicas) of the same tasks on different nodes, publishing their output on the same topics. A voter then selects the best results for further processing. Based on the methods described above, a total system failure due to a single node failure seems extremely unlikely. Note: This functionality is only implemented by high reliability requirements. Normal high availability systems may tolerate a restart. Figure 6 shows a possible launcher configuration.

 doi:10.20944/preprints202501.0857.v1

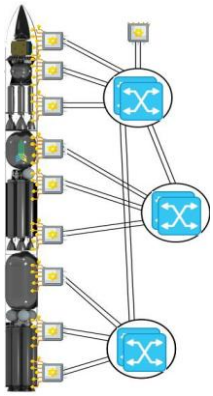

Figure 6. Possible launcher configuration.

6.2. Dual-X

For longevity, persistence, reliability, fault tolerance and error masking, all hardware is implemented as dual-X (X: controller, router, links, switches, etc.), meaning that each basic hardware unit is duplicated, as shown in Figure 7. The system consists of two parallel time-sensitive Ethernet networks [13]. Each controller, e.g. the dual board computer or the dual payload computer or the dual GNC computer, has two interfaces to the network, one for each router. Each Dual-X is therefore implemented with a total of four connections to the Dual network.

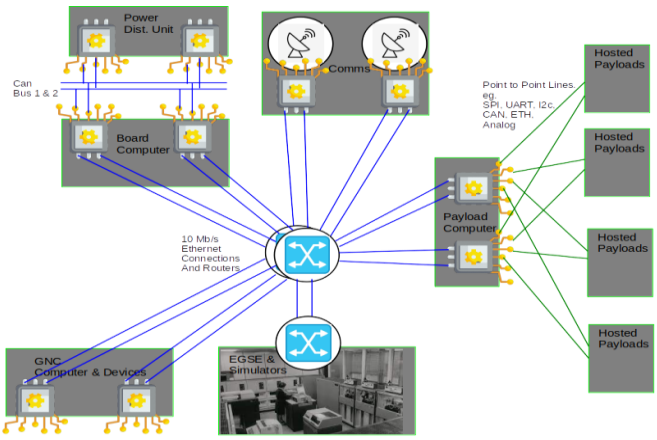

Figure 7. Dual-X structure of the avionics.

Beyond this dualisation, we can have more than one (dual) on-board computer (a matter of tailoring) and we can dynamically switch each pair on and off and then migrate, distribute and duplicate any software task and add voters at any time.

The cold, warm or hot redundancy arrangement of each pair is a software issue and can vary at different mission phases. For high reliability, e.g. for launchers, during manoeuvres or when engines are running, reliability must be guaranteed. For these situations, we recommend a hot-redundant configuration of computer and network pairs. For longevity over years, e.g. satellite operations, we recommend a cold redundancy configuration of compute and network pairs.

6.4. Computing Node

The computing nodes are implemented using powerful dual-core computers. The high performance core is used for applications, while the low performance core is used as an IO processor to implement time-triggered communication. High performance has two good sides. On the one hand, we can use the same computer as bus controller and payload computer at the same time, and transient failures are best detected and masked by using time redundancy [10]: run vital applications

 doi:10.20944/preprints202501.0857.v1

3 times in the same node and compare the results using voters. If all 3 applications are running in the same node, then we have no overhead for synchronisation. In the event of a total crash or hardware failure, we transfer control to a spare unit (in the same pair) and force a reboot and recovery from the failed unit. Whether the spare unit is hot or cold redundancy is a software issue and can vary according to mission requirements and phase.

Figure 8 shows the implementation of each dual computer (e.g. dual board computer, dual payload computer and dual GNC computer). There are 2 aspects to be explained: 1. IO adapter and 2. Keep-off circuit.

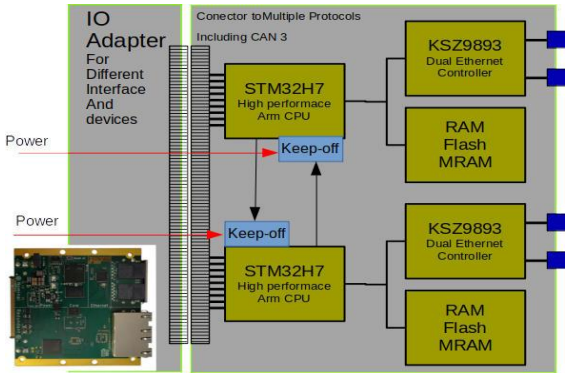

Figure 8. Dual Computer.

We have only one dual computer design which can be used for different functions and for different payloads in different missions. As there is only one standardized dual computer configuration, an IO adapter module will be used to implement the necessary interfaces for various payloads and mission types.

A key and central feature of our intelligent and highly versatile redundancy management is the keep-off circuit. Redundancy management is completely software controlled, allowing easy switching between hot and cold redundancy. The keep-off circuit is a switch that is part of the power distribution system. By default, the switch is closed, allowing power to flow. A pulse opens the switch for a configurable time, set by an RC (resistor-capacitor) [14] constant (e.g. 2 seconds). To keep the unit off, pulses must be sent at intervals shorter than this time, otherwise the switch closes and the unit switches on. As you can see in Figure 8, one computer, the Worker, can keep the other, the Spare, off as long as the Worker is running. For hot redundancy we do not send this keep off pulse. For cold redundancy, the worker keeps the spare computer off by sending this pulse periodically. If the pulse does not arrive on time, the spare computer is switched on and takes control (becomes the worker) and sends keep off pulses to the other (failed) computer, which is then switched off and becomes the spare computer.

Each computing node has an application called the watchdog. The watchdog checks the timing and correct behavior of all the other local applications. If everything seems to be working properly, it sends one pulse (per second) to the keep-off circuit of the other computer. If the worker crashes or the watchdog thinks something is very wrong, then the keep-off pulse will not be sent and the other computer will take over.

You may be thinking: With cold redundancy, both computers could turn each other off in a loop during the first power-up. To prevent this, there are two solutions: either assign different RC constants to each computer, or, as we did, use hardware random generators so that each computer waits a random amount of time before sending the first keep-off pulse.

6.5. CORFU

CORFU, an extensible model-driven framework for satellite software development, is being utilized for the first time in the InnoCube mission, scheduled for launch on January 14, 2025. This

 doi:10.20944/preprints202501.0857.v1

mission highlights the framework's flexibility and scalability in real-world space applications [15,16]. Concurrently, CORFU has been successfully adapted for the development of avionics and on-board software for HyMOVE. The versatile core avionics system benefits from the configurable and available architecture provided by CORFU [17].

Every application in the system communicates by publishing data to or subscribing to predefined core topics (see Figure 9). These topics act as shared channels for transmitting information such as telemetry, commands, system status and anomalies. This approach ensures modular and flexible communication, allowing seamless integration of new applications without requiring direct modifications to existing software components. This architecture simplifies integration and allows new functionalities to be added with minimal effort.The core applications (see Figure 10) are implemented with an initial basic configuration, designed to be flexible and easily adaptable to accommodate evolving requirements in the future. The graphical interface generated by CORFU enables both the visualization of telemetry data and the transmission of telecommands, providing a tool for both testing and mission operation phases. This functionality is facilitated through the "ground-connector" interface (see Figure 10), while telemetry data can also be concurrently stored in an InfluxDB database for further analysis and archiving.

The model-driven design simplifies the development of a reliable and above all scalable software system.

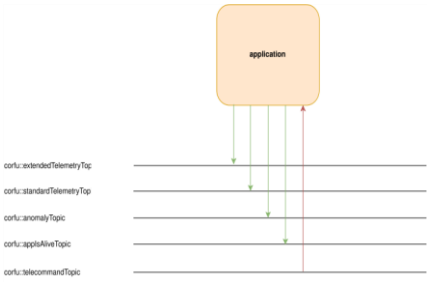

Figure 9. CORFU – core topics.

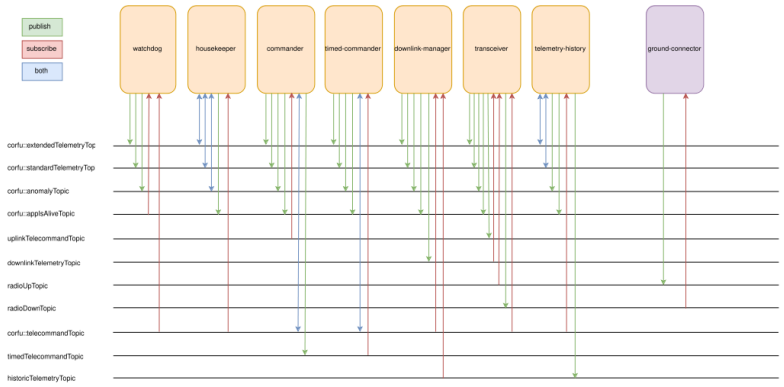

Figure 10. HyMOVE - core applications and topic access.

8. Conclusions

In conclusion, our approach to task migration, combined with the adaptable framework CORFU, presents a promising solution to enhance both reliability and flexibility in space mission operations.

This cobination can address challenges of modern, fast evolving space systems by ensuring continuous operation and adaptability, even when mission requirements or hardware constraints are changed

 doi:10.20944/preprints202501.0857.v1

References

1. Ferdinand Herte; et al. HyImpulse Small Launcher SL1 – Access to Space with Hybrid Propulsion. 35th Annual Small Satellite Conference, 2023. Available online: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=5035&context=smallsat (accessed on 07. January 2025).
2. Paola Breda; et al. Maiden flight of hyimpulse sr75 hybrid sounding rocket. 26th ESA Symposium on European Rocket and Balloon Programmes and related Research. Luzern, Switzerland, 2024. Available online: https://www.researchgate.net/publication/385073421_MAIDEN_FLIGHT_OF_HYIMPULSE_SR75_HYBRID_SOUNDING_ROCKET (accessed on 07. January 2025).
3. Paola Brenda; et al. HyMOVE: enabling HyImpulse in-orbit capabilities for small satellite missions. 75th International Astronautical Congress (IAC)At: Milan, Italy, 2024.
4. HyMOVE press release. Available online: https://hyimpulse.de/Press_Release/121224_Press%20Release_%20HyImpulse%20unveils%20HyMOVE.pdf (accessed on 07. January 2025).
5. RODOS. Available online: https://en.wikipedia.org/wiki/Rodos_(operating_system) (accessed on 07. January 2025).
6. Publish-subscribe pattern. Available online: https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern (accessed on 07. January 2025)
7. Dependability. Available online: https://en.wikipedia.org/wiki/Dependability (accessed on 07. January 2025).
8. Cold, warm and hot redundancy: determining how much you need. Available online: https://www.automationit.com/blog/79-cold-warm-and-hot-redundancy-determining-how-much-you-need (accessed on 07. January 2025).
9. Schneider Electric. What are the different types of Automation Redundancy?. Available online: https://www.se.com/eg/en/faqs/FA177620/ (accessed on 07. January 2025).
10. Sparsh Mittal; Subhhrajit Nag. A survey of encoding techniques for reducing data-movement. Journal of Systems Architecture. 2019. Available online: https://www.sciencedirect.com/topics/computer-science/time-redundancy (accessed on 07. January 2025).
11. ARINC653. Available online: https://en.wikipedia.org/wiki/ARINC_653 (accessed on 07. January 2025).
12. IEEE Standard for Local and Metropolitan Area Networks--Timing and Synchronization for Time-Sensitive Applications Amendment 1: Inclusive Terminology. Available online: https://standards.ieee.org/ieee/802.1ASdr/10568/ (accessed on 07. January 2025).
13. Time-Sensitive Networking. Available online: https://en.wikipedia.org/wiki/Time-Sensitive_Networking (accessed on 07. January 2025).
14. RC time constant. Available online: https://en.wikipedia.org/wiki/RC_time_constant (accessed on 07. January 2025).
15. Frank Flederer. CORFU - An Extended Model-Driven Framework for Small Satellite Software with Code Feedback. doctoralthesis, University of Wuerzburg, 2021. Available online: https://opus.bibliothek.uni-wuerzburg.de/opus4-wuerzburg/frontdoor/deliver/index/docId/24981/file/Flederer_Frank_Dissertation.pdf (accessed on 07. January 2025).
16. Tom Baumann, Erik Dilger, Sergio Montenegro, Felix Sittner, Michael Strohmeier, Thomas Walter: InnoCube - Preparing the Fully Wireless Satellite Data Bus for Launch, University of Wuerzburg, SSC23-WIV-03. Available online: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=5556&context=smallsat (accessed on 07. January 2025).
17. Frank Flederer; Sergio Montenegro. A configurable framework for satellite software. In 2021 IEEE 12th International Conference on Software Engineering and Service Science (ICSESS), 2021; pages 28–31.

 doi:10.20944/preprints202501.0857.v1