Catalysing Next-Generation SmallSat Missions with Scalable Model-Based Flight and Payload Software

Conrad Gillespie, Bright Ascension Ltd., Edinburgh, UK

Abstract

Small satellite missions are increasingly constrained not by hardware capability, but by the cost, risk, and schedule friction associated with developing mission-critical flight and payload software. While payload computers and onboard processing hardware have become highly commoditised, the software required to make these systems flight-ready remains complex, engineering-intensive, and frequently bespoke. Teams are repeatedly required to implement foundational functionality - secure communications, data handling, automation, telemetry, and fault management - before any mission-specific value can be realised.

This paper presents a scalable, model-based and service-oriented approach that addresses this challenge, through the combined use of HELIX Edge, a mission-ready payload software product, and HELIX Appkit, a unified framework for the development and lifecycle management of payload applications. The approach enables earlier payload computer utilisation, reduced integration risk, improved testability, and the ability to evolve mission capability post-launch. The paper focuses on architectural principles, development and integration workflows, and lessons learned from representative nano and microsatellite payload deployments.

Keywords
Small satellites, payload software, flight software, model-based engineering, service-oriented architecture, edge computing, onboard processing.

1. Introduction

The small satellite sector has undergone a period of rapid transformation over the last decade. Advances in launch availability, standardised satellite form factors, and high-performance commercial processors have dramatically reduced the barrier to entry for space missions. Payload computers that once required custom radiation-tolerant designs are now routinely delivered as configurable, FPGA-based, Linux-capable systems intended to support complex onboard processing workloads.

Despite this progress, payload software development has emerged as a dominant driver of cost and schedule risk. Payload computers are typically supplied with minimal board support packages and limited reference software, leaving mission teams to design and implement the majority of the flight-critical software stack. As missions demand greater onboard autonomy, including real-time processing, data reduction, and adaptive operations, the gap between hardware readiness and software readiness has widened.

This paper argues that small satellite missions require a shift away from fully bespoke payload software towards configurable, mission-ready software products supported by extensible application frameworks. By stabilising the software foundation early and isolating mission-specific logic, teams can accelerate development while improving reliability and long-term maintainability.

2. Background and Motivation

Payload software performs a dual role within small satellite systems. It must provide reliable, deterministic infrastructure service - such as command handling, telemetry generation, and fault detection - while simultaneously supporting highly mission-specific processing algorithms. In many projects, these concerns are tightly coupled, resulting in monolithic applications that are difficult to validate, extend, or reuse.

In parallel, small satellite missions increasingly emphasise responsiveness to changing requirements. Missions may wish to deploy new onboard processing algorithms, modify automation logic, or adjust operational behaviour based on on-orbit experience. Traditional development approaches, in which payload software is frozen early to mitigate risk, are poorly suited to this operational reality.

Model-based engineering and service-oriented architectures provide mechanisms to manage this complexity. By defining system behaviour declaratively and using well-defined service interfaces, it becomes possible to decouple software infrastructure from mission logic. The HELIX approach builds on these principles, with an explicit focus on deployability, operational integration, and lifecycle evolution.

3. HELIX Software Architecture Overview

The HELIX ecosystem is founded on a common core technology used across flight software, ground systems, simulation environments, and application frameworks. At the centre of this approach is a core technology which is model-driven, component-based and service-oriented. The use of this technology means that HELIX products are highly modular, use a machine-readable model to provide an "understanding" of the system, and integrate through well-defined, standard interfaces. By sharing the HELIX technology at their core, HELIX products can work together seamlessly, forming an ecosystem.

Within this ecosystem, HELIX distinguishes between mission-ready products, intended for immediate operational use, and component-based development kits that enable extension and customisation. HELIX Edge and HELIX Appkit together form the payload-side implementation of this philosophy with other products, such as HELIX Ops, on the ground side.

3.1 HELIX Technology

The HELIX core technology is Model-driven, Service-oriented, and Component-based:

- **Model-driven**: A HELIX model describes the software systems involved in a mission. It defines the capabilities and configuration of each system and how they can be used by others, as well as capturing how the various systems are connected.
- **Component-based**: The building blocks of all HELIX software are components which represent a self-contained and coherent set of functionality.
- **Service-oriented**: The components in a HELIX system use services to interact, which provide well-defined syntax and semantics for interactions.

Models

Models are the most fundamental aspect of HELIX, the foundation that everything else builds on. HELIX models are machine-readable, which means they can be used across the full mission life-cycle. HELIX development kits, such as Appkit, are built around them, automating large parts of the software development workflow. HELIX application products, such as Edge, use them to understand and interact with other elements in the wider space system. For example, the HELIX model-based approach allows the payload applications, developed through our HELIX Appkit, to be quickly and easily understood by other parts of the system, including HELIX Ops to operate the payload applications in the context of the mission. This gives Ops a full view of the different components and their services, enabling easy interaction at a very high level and providing powerful opportunities for automated operations. The sharing of models makes integration and configuration virtually automatic.

Components

Components are one of the key aspects that allow HELIX to realise the flexibility offered by the model-driven approach. Being component-based means software is either constructed from, or described as being made of, well-defined self-contained elements, which are flexible and intrinsically reusable.

While every mission is different, all mission software systems on many missions perform similar tasks such as data acquisition, monitoring, logging. Despite being highly mission-specific, payload applications frequently perform many similar activities, such as payload sequencing and orchestration, data product

management and processing using common algorithms. Within HELIX development kit, including Appkit, common functions are provided as library Components to accelerate the development of software. Mission-specific software systems are developed by creating any necessary bespoke components and combining them with previously validated library components using the mission model. This is supported by code generation tools to provide a high-speed development workflow.

Services

Services are the other key aspect that allows HELIX to realise the flexibility offered by our model-driven approach. A service-oriented approach means that specific components, and the parts of the system built from them, don't have to be hard-linked to other components, allowing them to be combined in many different ways depending on the needs of individual missions.

Services describe the facilities and behaviour that components either provide or require. A component that provides a service is promising to behave in a certain way, and so a component that uses that service knows exactly what to expect, despite not knowing which specific component realises that behaviour or where it is located. This provides a flexible way of integrating software components, both within a software system, such as a payload application built using Appkit, and between software systems, such as between the payload Application and Edge or between Edge and ground systems. That flexibility means that as the space system grows or a need for new missions emerges, existing components, apps, and even whole systems can be swapped in and out and HELIX can be quickly adapted to a new scenario.

3.2 HELIX Models and HELIX Apps

A HELIX model for a given mission is a single machine-readable "source of truth" which describes all aspects of the mission's systems, including the structure of the various mission hardware and software systems and the interfaces between them. Although models are usually used to describe the whole mission, and that is how we will describe them here, they can also be applied to part of a system such as a payload computer and its applications.

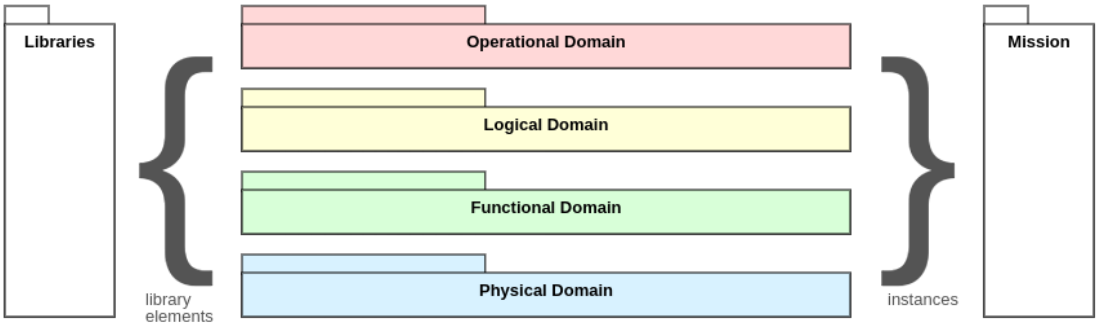
As described above, the use of models is central to the HELIX approach. Models are the most important of the three pillars of HELIX: the model-driven workflow is what allows HELIX to use components and services so effectively.

The information captured in a mission model is used for many different purposes, by lots of different actors and tools, throughout all stages of a mission lifecycle. For example:

- In the hardware design phase, HELIX tooling could provide access to information from the model to facilitate the calculation/assessment of mass budgets and so on.
- In software development, HELIX tooling such as Appkit uses the information in the model to generate code which connects separate software components to each other via the services they each provide and consume.
- The model drives test and development environments, intrinsically supporting early integration and test.
- During operational use, HELIX products use the mission model to determine the capabilities and configuration of each spacecraft, and to know how to talk to them.
- Documentation is tool-generated from a model, this includes design documentation and information for important actors such as spacecraft operators.

Fundamentally, the HELIX model supports communications, collaboration, and iteration between stakeholders: from data consumers and operators to developers and testers.

A model can describe a mission at several levels, from the hardware level up through increasingly high levels of abstraction. HELIX calls these separate levels model domains. Each domain can access and build on elements from the domains below it.



- The physical domain describes the presence and construction of hardware devices.
- The functional domain describes the software elements that are available.
- The logical domain describes logical views of the services and interface elements available, distinct from the functional and physical structures of the software and hardware.
- The operational domain describes the facilities used by ground software and spacecraft operators for longer-term control of spacecraft.

The functional domain is used by developers with Appkit to describe the components in the payload applications they are building. The model produced by Appkit for a payload app can be easily combined with the model of HELIX Edge to form a complete model of the payload. Changes in app availability onboard the payload computer can be easily reflected in model updates. As each update to the model is versioned, the model is not only able to reflect the state of the system now, but also prior states.

Whereas the functional domain describes what the system can do, the operational domain complements this by describing what *should* be done with the system in the form of reusable operational procedures. A HELIX app developed with Appkit is therefore incredibly powerful as it not only contains the app executable itself, but also a full description of its functions and operations procedures in the form of a model, together with embedded human-readable documentation.

4. HELIX Edge: Mission-Ready Payload Software

HELIX Edge is designed to address the recurring foundational requirements of payload computers used on small satellite missions. Delivered as a pre-integrated and validated software image for specific hardware platforms, Edge allows payload computers to be powered on and used immediately within an integration environment.

At its core, Edge provides a payload data handling service responsible for routing packets between the spacecraft platform, onboard payload applications, and ground systems. This service enforces well-defined interfaces, manages telemetry and logging, and implements event-driven automation policies. Security mechanisms, including authenticated command handling and encrypted communications where required, are incorporated at the infrastructure level rather than left to individual application developers.

By embedding these capabilities in a mission-ready baseline, Edge significantly reduces the amount of bespoke code required for each mission. Teams can concentrate effort on payload-specific processing while relying on a consistent, flight-proven foundation for infrastructure concerns.

5. Payload Computer Integration Strategy

HELIX Edge supports a set of widely available, industry leading payload computers, with the range of supported computers growing all the time. Payload computers supported by HELIX Edge are Linux systems, often FPGA-based, and are typically delivered with a vendor-specific build environment utilising Yocto. Edge is provided as a prepackaged binary which can be pre-installed on a payload computer providing a rapid-start out-of-the-box experience by utilising the graphical test client shipped as part of Edge. In addition,

the Edge environment is provided as a Yocto layer which can be used as the starting point for reconfiguration of the Linux environment and full customisation of the flight condition of the payload computer.

In practice, significant variability exists between individual configurations of even ostensibly identical hardware platforms. Edge addresses this through an opinionated integration strategy that limits supported configurations while allowing controlled adaptation.

The key features of HELIX Edge cover the following responsibilities:

- Interfacing to the platform, and therefore to ground, including security, with support for file transfer and file system management. Synchronisation of payload computer time to platform time and PPS is also supported.
- Interfacing to payload software applications both at the raw packet level and at the level of HELIX services.
- Management of execution of payload software applications including HELIX Apps built with Appkit.
- Management of the underlying payload computer including watchdogs and telemetry reporting.
- Handling of telemetry, including reporting in beacons, logging and monitoring.
- Automation of operations through time-based scheduling and event-based automation.
- Safe management of flight software images, both at the level of the payload computer boot image, using computer-specific features, and at the level of the core flight software applications making up Edge.

The HELIX Edge approach balances repeatability with flexibility, allowing rapid support for new missions while maintaining a high level of confidence in software behaviour.

6. HELIX Appkit: Payload Application Framework

HELIX Appkit complements Edge by providing a structured environment for developing and deploying payload applications from reusable software components. Rather than embedding mission logic directly within the payload data handler, application developers implement discrete services that interact with Edge through standardised interfaces.

Applications developed using Appkit benefit from a consistent execution model, managed lifecycle, and shared integration with the mission model. Edge is responsible for starting, stopping, monitoring, and isolating applications, reducing the likelihood that faults in mission logic propagate to core payload functions. Where supported by the hardware platform, applications may be containerised, further improving isolation and deployment flexibility.

7. Development, Test, and Operations Workflow

The combined Edge and Appkit approach supports an iterative development workflow aligned with modern software practices. Payload applications can be developed and tested against representative Edge environments prior to full spacecraft integration. Mission models used during development are directly reused during operations, supporting high-fidelity testing and automated procedure execution.

Following launch, new applications or updated versions of existing applications may be deployed within the constraints defined by mission assurance processes. This enables missions to adapt operational behaviour based on real-world performance while avoiding the risks associated with modifying core flight software.

8. Discussion and Lessons Learned

Experience across multiple payload integration activities highlights several recurring benefits of the HELIX approach. Early availability of a stable software foundation significantly improves the efficiency of

hardware-in-the-loop testing and reduces late-stage integration surprises. Clear separation between infrastructure and mission logic improves both verification and maintainability.

The HELIX model forms a powerful core for managing both mission-generic and mission-specific functions across the complete development life-cycle, from rapid development to test and flight deployment. The primary trade-off lies in the upfront discipline required to capture information in the model. In our experience, this investment is rapidly recovered through reduced debugging effort, improved reuse, and greater operational confidence.

9. Conclusion

As small satellite missions continue to increase in complexity and ambition, software infrastructure must evolve accordingly. HELIX Edge and HELIX Appkit demonstrate that payload software can be treated as a configurable product supported by extensible applications, rather than re-engineered for each mission. A model-based approach provides support for the development and long-term operations lifecycle, capturing critical information in a machine-readable form which can be accessed by tools and applications and which acts as a single source of truth.

By combining mission-ready software with model-based extensibility, this approach enables faster development, lower risk, and long-term adaptability—key enablers for next-generation small satellite missions operating in increasingly dynamic environments.