NASA NOS3: Integration Analysis for Ground Station Modernization

Wesley Robbins

STSGym Research • Ground Mission Systems Division

April 2026

Abstract. *NASA's Operational Simulator for Space Systems (NOS3) provides a comprehensive, open-source spacecraft simulation platform developed at the Katherine Johnson Independent Verification and Validation Facility. This paper presents a detailed architectural analysis of NOS3 across its three primary layers—ground software, flight software, and simulation—and evaluates its suitability for integration with modern ground mission system ecosystems. We examine four ground software options (COSMOS, OpenC3, YAMCS, F Prime GSW), two flight software frameworks (cFS and F Prime), and the combined 42 Dynamics and NOS Engine simulation backends. The communication architecture based on UDP socket interfaces and CCSDS-compliant protocols is analyzed in depth, including the CFDP file delivery protocol and generic radio component model. We present a phased integration roadmap for converting NOS3's simulation capabilities into an operational ground station platform, covering protocol bridging, hardware-in-the-loop validation, and real RF chain deployment. Comparisons with alternative simulation frameworks (STK, GMAT, OpenSatKit) position NOS3 as a uniquely accessible, full-stack spacecraft development environment. The paper concludes with recommendations for real hardware bridging and operational deployment paths.*

Table of Contents

1. Introduction
2. NOS3 Architecture Overview
3. Ground Software Analysis
4. Flight Software Frameworks
5. Simulation Engine
6. Communication Architecture
7. Component Simulator Library
8. Integration Roadmap
9. Build System & Deployment
10. Related Work
11. Conclusion & Future Work
12. References

---

1. Introduction

NASA's investment in spacecraft simulation spans decades, from early hardware-in-the-loop test benches to modern software-only environments that can replicate entire missions on a developer's laptop. The Operational Simulator for Space Systems (NOS3) represents the current state of this evolution: a complete, open-source spacecraft simulation platform that integrates ground software, flight software, dynamics modeling, and hardware component simulation into a single coherent system.

The motivation for this analysis arises from a practical need. As ground station technology moves from monolithic, facility-bound installations toward distributed, containerized architectures, the question of how simulation platforms like NOS3 bridge to operational

ground systems becomes critical. Traditional ground stations assume dedicated RF hardware, fixed infrastructure, and proprietary software stacks. Modern ground mission systems (GMS) assume cloud-native deployment, API-first integration, and multi-mission flexibility.

NOS3 occupies a unique position in this transition. It is simulation-grade—designed for verification and validation—yet its architecture maps directly onto operational patterns. Its UDP-based generic radio component mirrors real RF link interfaces. Its cFS flight software stack runs on the same core that flies on NASA missions. Its YAMCS ground software backend is the same framework used by ESA and commercial operators for mission control.

This paper provides a comprehensive architectural analysis of NOS3 and presents an integration roadmap for converting its simulation capabilities into an operational ground station platform. We examine each layer of the NOS3 stack, analyze the communication protocols that bind them, and propose a phased approach to bridging simulation to reality.

2. NOS3 Architecture Overview

NOS3 is organized into three distinct layers that communicate through well-defined interfaces. The ground software layer (GSW) provides operator interfaces for commanding and telemetry. The flight software layer (FSW) implements the onboard spacecraft logic. The simulation layer provides orbital dynamics and hardware component models. A generic radio component mediates all communication between GSW and FSW through UDP sockets.

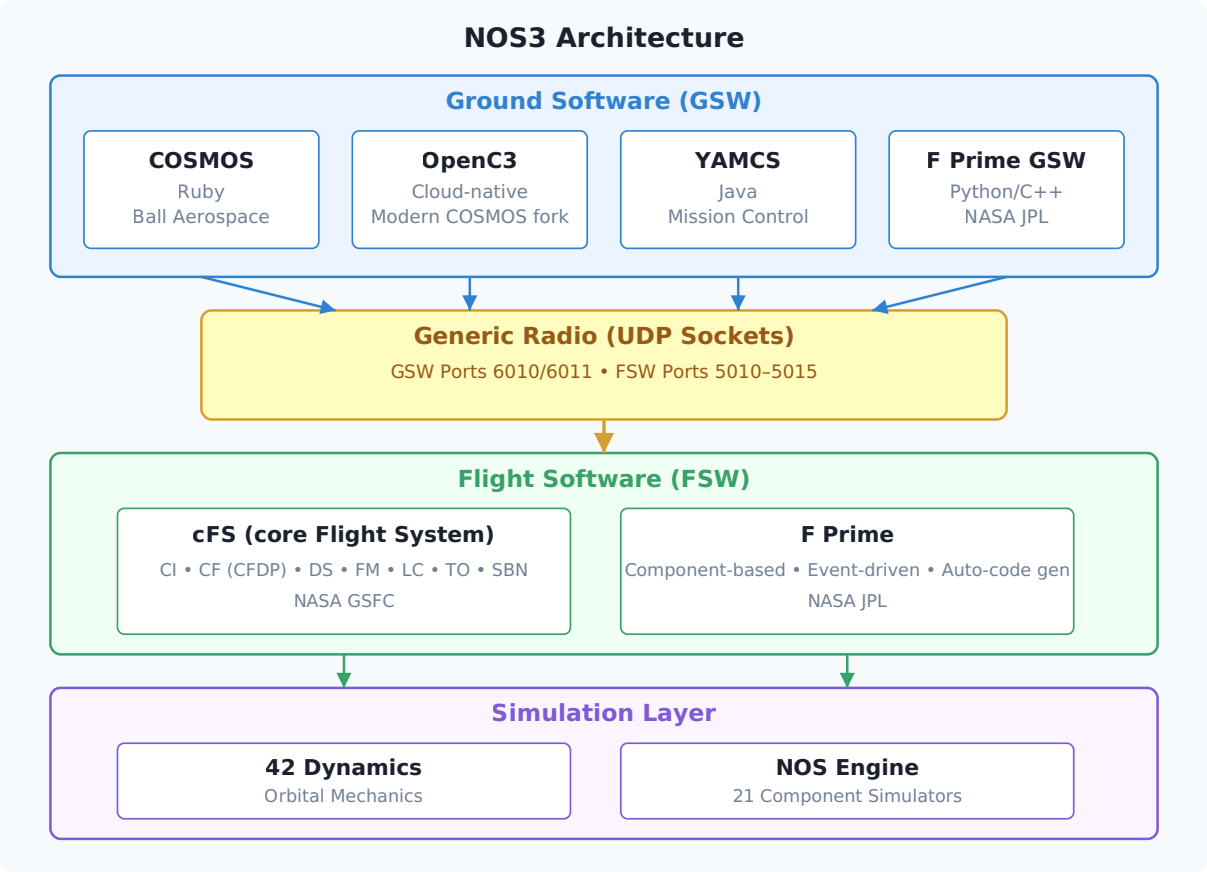

Figure 1: NOS3 three-layer architecture showing ground software, generic radio communication interface, flight software, and simulation components.

The architecture's key design principle is layer separation. Each layer can be independently configured, replaced, or upgraded without affecting the others. The mission configuration file (`nos3-mission.xml`) specifies which GSW, FSW, and scenario to compose, enabling rapid reconfiguration between development, test, and mission profiles.

3. Ground Software Analysis

NOS3 provides four ground software options, each targeting different operational needs. This section analyzes each option's architecture, strengths, and integration potential.

3.1 COSMOS

COSMOS, developed by Ball Aerospace (now part of BAE Systems), is the default ground software in NOS3. Built on Ruby, it provides a rich GUI for command and telemetry operations, script automation via the Ruby scripting interface, real-time data visualization, and a library of built-in test procedures. COSMOS deploys as a Docker container and is launched via the NOS3 Makefile target `make cosmos-operator`.

COSMOS excels in rapid prototyping and operator familiarity. Its Ruby scripting allows operators to automate complex command sequences without compilation. However, its monolithic architecture and limited cloud deployment options constrain its use in distributed ground station environments.

3.2 YAMCS

YAMCS, developed by Space Applications Services, is a mission control framework built in Java with a web-based interface accessible via standard browsers. It is CCSDS-compliant, provides time-correlated data management with archive and replay capabilities, and integrates with OpenMCT for advanced visualization. YAMCS is fully integrated in NOS3 and accessible through a web interface.

YAMCS represents the strongest candidate for ground station modernization. Its web-native architecture eliminates the need for client-side installations. Its CCSDS compliance ensures interoperability with standard ground station equipment. Its REST API enables programmatic integration with external systems. The AGPLv3 license is acceptable for research and many operational contexts.

3.3 OpenC3

OpenC3 is a cloud-native fork of COSMOS that modernizes the architecture for Kubernetes deployment. It retains COSMOS's Ruby/JavaScript foundation while adding enterprise features: scalable microservice architecture, container orchestration support, and a modern web UI. OpenC3 targets organizations running ground stations in cloud environments where horizontal scaling and containerized deployment are primary concerns.

3.4 F Prime Ground Software

F Prime's ground software, developed at NASA JPL, is a Python/C++ application that provides command and telemetry interfaces specifically designed for F Prime flight software. It includes the F Prime Ground System with event-driven data processing, automatic code generation for command dictionaries, and tight integration with F Prime's component architecture. While it lacks the general-purpose telemetry processing of YAMCS or COSMOS, it provides the most seamless experience for F Prime-based missions.

COSMOS

- **Strengths:** Mature, rich GUI, Ruby scripting
- **Limitations:** Monolithic, desktop-bound
- **Best for:** Rapid prototyping, traditional ops

YAMCS

- **Strengths:** Web-native, CCSDS, API-first
- **Limitations:** JVM overhead, AGPL license
- **Best for:** Modern ground stations, multi-mission

OpenC3

- **Strengths:** Cloud-native, Kubernetes-ready
- **Limitations:** Enterprise pricing, less mature
- **Best for:** Cloud-deployed ground stations

F Prime GSW

- **Strengths:** Tight F Prime integration, auto-code gen
- **Limitations:** F Prime-only, narrow scope
- **Best for:** JPL-style component missions

4. Flight Software Frameworks

NOS3 supports two flight software frameworks: the core Flight System (cFS) from NASA Goddard Space Flight Center and F Prime from NASA's Jet Propulsion Laboratory. Both are open-source, flight-qualified frameworks, but they embody fundamentally different architectural philosophies.

4.1 cFS Architecture

The core Flight System is a modular, message-passing framework built around a software bus paradigm. Applications communicate by publishing and subscribing to messages on the Software Bus Network (SBN). The cFS core consists of the following applications:

- **CI (Checkout)**: Provides hardware checkout commands during integration and test.
- **CF (CFDP)**: Implements the CCSDS File Delivery Protocol for reliable file transfer between ground and spacecraft.
- **DS (Data Store)**: Manages onboard data storage and downlink prioritization.
- **FM (File Manager)**: Provides file system operations for onboard mass storage.
- **LC (Limit Checker)**: Monitors telemetry against defined thresholds and generates alerts.
- **TO (Telemetry Output)**: Formats and routes telemetry packets for downlink.
- **SBN (Software Bus Network)**: Provides the publish/subscribe messaging backbone.

cFS applications are compiled as shared libraries loaded by the cFE (core Flight Executive) at startup. This architecture allows mission-specific applications to be added without modifying the core framework. The software bus decouples applications at the message level, enabling independent development and testing.

4.2 F Prime Architecture

F Prime takes a component-based, event-driven approach. Unlike cFS's message bus, F Prime components communicate through typed ports with explicit input/output connections defined in the topology. Key characteristics include:

- **Component model:** Each component has typed ports (commands, telemetry, events, parameters) defined in a domain-specific language.

- **Auto-code generation:** F Prime's build system generates serialization, command handling, and telemetry reporting code from component definitions.
- **Topology definition:** Component instances and their interconnections are declared in topology files, enabling static analysis of data flow.
- **Event-driven execution:** Components respond to asynchronous events through port invocations rather than polling.

4.3 Framework Comparison

Aspect	cFS	F Prime
Architecture	Message bus (pub/sub)	Component ports (typed connections)
Communication	Software Bus Network	Port-based topology
Code Generation	Manual / script-assisted	Automatic from definitions
Coupling	Loose (message-level)	Tight (port-level, compile-time)
Debugging	Message inspection on bus	Event logging + port tracing
Heritage	NASA GSFC missions	NASA JPL missions (Mars Helicopter)
Language	C	C++

The choice between cFS and F Prime depends on mission philosophy. cFS's loose coupling and message bus favor flexibility and late integration. F Prime's typed ports and auto-code generation favor early validation and correctness by construction. NOS3's support for both allows teams to evaluate each approach within the same simulation environment.

5. Simulation Engine

The NOS3 simulation layer combines two complementary engines: 42 Dynamics for orbital mechanics and the NOS Engine for hardware component simulation.

5.1 42 Dynamics Engine

42 is NASA's general-purpose spacecraft dynamics simulator that provides high-fidelity orbital propagation, attitude dynamics, and environmental modeling. Given initial conditions derived from Two-Line Element (TLE) sets, 42 propagates the spacecraft orbit using numerical integration of the equations of motion.

The fundamental orbital propagation follows Kepler's equation. For a spacecraft with semi-major axis a, eccentricity e, and mean motion n, the mean anomaly at time t is:

$$M(t) = M_0 + n(t - t_0)$$

Kepler's equation relates the mean anomaly M to the eccentric anomaly E:

$$M = E - e \sin E$$

This transcendental equation is solved iteratively (typically via Newton-Raphson). From the eccentric anomaly, the true anomaly ν is computed as:

$$\tan\frac{\nu}{2} = \sqrt{\frac{1+e}{1-e}} \tan\frac{E}{2}$$

The radial distance follows:

$$r = a(1 - e \cos E)$$

42 extends these basic calculations with perturbation models including J2 oblateness, atmospheric drag, solar radiation pressure, and third-body gravitational effects. The simulator outputs spacecraft position, velocity, attitude, and angular rates at configurable time steps, providing the dynamic state that drives NOS Engine component models.

5.2 NOS Engine

NOS Engine provides the simulation backbone for 21 hardware component simulators. It operates as a time-synchronized simulation bus that connects component models to the flight software through the same interfaces used by real hardware. Components register with NOS Engine, which manages timing synchronization, data flow, and the simulation clock.

The architecture supports hardware-in-the-loop (HITL) testing: real hardware can replace simulated components by connecting through the same NOS Engine interface. This enables progressive validation from pure software simulation through hybrid simulation with selected real hardware to full hardware integration.

5.3 Hardware-in-the-Loop Strategy

NOS3's simulation architecture supports a three-phase validation progression:

1. **Pure software simulation:** All components run as NOS Engine models. The 42 Dynamics engine provides orbital state. This is the default NOS3 configuration.
2. **Hybrid HITL:** Selected hardware components (e.g., a real reaction wheel or star tracker) are connected to the NOS Engine bus, replacing their simulated counterparts. Flight software interacts with real hardware through the same interface.
3. **Full hardware integration:** The flight software runs on target hardware (e.g., a radiation-hardened processor), with real components connected through actual hardware interfaces. NOS Engine provides only the dynamics simulation.

This progression is critical for ground station conversion because it provides a validated path from simulation to operations. The same software that is verified in simulation runs unchanged on real hardware.

6. Communication Architecture

The NOS3 communication architecture is built on UDP socket interfaces that model the radio link between ground and spacecraft. This section analyzes the protocol stack, the generic radio component, and CCSDS standards compliance.

6.1 UDP Socket Interface

The generic radio component mediates all communication between GSW and FSW through UDP sockets on defined port numbers. This design choice has significant implications:

- **Simulation fidelity:** UDP mirrors the lossy, unordered nature of space radio links more closely than TCP, enabling realistic testing of packet loss and reordering scenarios.

- **Decoupling:** UDP socket interfaces allow GSW and FSW to run in separate processes or containers, matching operational deployment patterns.
- **Protocol independence:** The UDP transport carries CCSDS-formatted packets, allowing higher-level protocol implementations (CFDP, etc.) to be tested independently of the transport layer.

Interface	Port	Direction
GSW → Radio	6010	Command uplink
Radio → GSW	6011	Telemetry downlink
FSW → Radio (Primary)	5010	Telemetry upload to radio
Radio (Primary) → FSW	5011	Command delivery to FSW
FSW → Radio (Proximity)	7010	Proximity link transmit
Radio (Proximity) → FSW	7011	Proximity link receive

6.2 Command Format

Commands to the generic radio follow a fixed 16-byte format with magic number validation:

```
 /* Generic Radio Command Packet (16 bytes) */
typedef struct {
    uint16_t magic1;      /* 0xDEAD at offset 0  */
    uint8_t  command_id;   /* Command ID at offset 2 */
    uint8_t  _pad;         /* Padding at offset 3    */
    uint32_t payload;      /* 4-byte payload at offset 4 */
    uint16_t magic2;       /* 0xBEEF at offset 8  */
    uint16_t _reserved;    /* Reserved at offset 10  */
    uint32_t _padding;     /* Pad to 16 bytes       */
} radio_cmd_t;
```

The dual magic number pattern (0xDEAD/0xBEEF) provides basic packet integrity checking at the radio interface level, supplementing the CCSDS frame-level checksums.

6.3 CFDP Protocol

The CCSDS File Delivery Protocol (CFDP), implemented in the cFS CF application, provides reliable file transfer over the inherently unreliable space link. CFDP operates in two modes:

- **Class 1 (Unreliable):** Best-effort file transfer with no retransmission. Suitable for telemetry data where occasional gaps are acceptable.
- **Class 2 (Reliable):** Acknowledged file transfer with negative-acknowledge (NAK) based retransmission. Suitable for command uplink and critical data downlink.

CFDP's NAK-based approach is optimized for the long-delay, high-error-rate space channel. Rather than acknowledging every packet (which would require excessive round-trip waits), the receiver sends a NAK only for missing data after the transfer completes or after detected gaps. This reduces protocol overhead on the space link by orders of magnitude compared to TCP-style acknowledgment.

6.4 CCSDS Standards Compliance

NOS3's communication stack implements CCSDS standards at multiple levels:

- **CCSDS 102.0:** Packet Telemetry—telemetry packets follow the standard primary header format with application process IDs.
- **CCSDS 202.0:** Telecommand—command packets use the standard telecommand frame structure.
- **CCSDS 727.0:** CFDP—the file delivery protocol follows the standard specification for both Class 1 and Class 2 operation.
- **CCSDS 133.0:** Space Packet Protocol—the underlying packet structure uses standard space packet headers.

This standards compliance is essential for ground station integration. Any CCSDS-compliant ground station equipment can receive and process NOS3 telemetry without custom protocol development.

7. Component Simulator Library

NOS3 includes 21 hardware component simulators that model the spacecraft's physical subsystems. These simulators run under NOS Engine and provide realistic responses to flight software commands.

Component	Subsystem	Description
generic_radio	Communication	UDP socket-based radio interface
generic_adcs	ADCS	Attitude determination and control system
generic_css	ADCS	Coarse sun sensor array
generic_eps	Power	Electrical power system with battery model
generic_fss	ADCS	Fine sun sensor
generic_imu	ADCS	Inertial measurement unit (gyro + accel)
generic_mag	ADCS	Three-axis magnetometer
generic_reaction_wheel	ADCS	Reaction wheel with torque/speed model
generic_star_tracker	ADCS	Star tracker with quaternion output
generic_thruster	Propulsion	Thruster with impulse model
generic_torquer	ADCS	Magnetic torque rod
arducam	Payload	Camera payload simulator
novatel_oem615	GNSS	GPS receiver (NovAtel OEM615)
cryptolib	Security	Cryptographic library for secure comm
blackboard	Data	Shared memory interface
sample	Template	Component development template
mgr	System	Simulation manager
syn	Time	Time synchronizer
onair	AI	On-orbit AI inference engine

The ADCS components collectively form a complete attitude determination and control simulation chain: sun sensors (CSS, FSS) provide coarse and fine sun vector measurements, the IMU provides angular rates and linear acceleration, the magnetometer measures the local magnetic field, and the star tracker provides high-accuracy attitude quaternions. The ADCS algorithms in the flight software fuse these sensor inputs to compute actuator commands for reaction wheels, magnetic torquers, and thrusters.

The power system simulator (generic_eps) models battery state-of-charge, solar array power generation (coupled to the ADCS attitude model), and power distribution. This creates realistic power constraints: an ADCS maneuver that points the solar arrays away from the sun will degrade the power budget, potentially triggering low-voltage conditions in the simulation.

The component template (`sample`) provides a documented starting point for adding custom components. Teams implementing specialized payloads or custom hardware interfaces can clone the template and implement the NOS Engine simulation interface without modifying the core framework.

8. Integration Roadmap

Converting NOS3 from a simulation platform to an operational ground station requires a phased approach that progressively replaces simulated components with real infrastructure while preserving the validated software stack.

> ***Phase 1: YAMCS Backend Deployment (Weeks 1–2)***
>
> Deploy YAMCS as the ground station backend with its REST API exposed to the GMS frontend. Configure YAMCS to receive CCSDS-formatted telemetry from the NOS3 simulation and establish command interfaces through the YAMCS command queue. Validate end-to-end data flow: NOS3 simulation → Generic Radio → YAMCS → GMS UI.

Phase 2: Protocol Bridge Service (Weeks 3–4)

Develop a protocol translator that maps the Generic Radio UDP interface to the GMS REST API. This service bridges NOS3's simulation-oriented UDP sockets to API-driven ground station services. Implement bidirectional translation: GMS commands are packaged as Generic Radio command packets and injected on the uplink port; telemetry received from the downlink port is parsed and forwarded to GMS services.

```
 # Protocol bridge configuration
bridge:
  udp_listen_port: 6011     # Radio → GSW telemetry
  udp_send_port: 6010       # GSW → Radio commands
  gms_api_endpoint: "https://gms.example.com/api/v1"
  ccsds_apid_mapping:
    0x01: "adcs_telemetry"
    0x02: "eps_telemetry"
    0x42: "payload_data"
```

Phase 3: Hardware Integration (Weeks 5–7)

Replace the generic radio simulation with a real SDR (Software Defined Radio) interface. The SDR provides RF modulation/demodulation while maintaining the same CCSDS packet structure. Test with simulated satellite signals first (SDR transmitting to itself via loopback or attenuated path), then progress to live satellite passes. Validate that the command and telemetry flows observed in simulation are reproduced with real RF.

> ***Phase 4: Visualization and Operations (Weeks 8–9)***
>
> Deploy OpenMCT for GMS dashboards with custom telemetry displays, ground track visualization, and pass planning tools. Integrate the YAMCS archive for historical data replay and anomaly investigation. Establish operational procedures for satellite pass support including pre-pass checkout, real-time monitoring, and post-pass data analysis.

9. Build System & Deployment

NOS3 uses a CMake-based, multi-target build system orchestrated through a top-level Makefile with over 50 targets. The build system supports cross-compilation for different target architectures and containerized deployment via Docker.

9.1 CMake Configuration

The mission configuration file drives the build:

```
 <nos3-mission-cfg>
    <start-time>814254200.0</start-time>
    <gsw>cosmos</gsw>
    <fsw>cfs</fsw>
    <scenario>STF1</scenario>
    <number-spacecraft>1</number-spacecraft>
</nos3-mission-cfg>
```

This XML configuration specifies the simulation start time (in J2000 epoch seconds), ground software selection, flight software framework, mission scenario, and spacecraft count. The CMake build system generates target-specific configurations from this mission definition.

9.2 Key Build Targets

Target	Description
`make prep`	Prepare development environment (fetch dependencies)
`make config`	Configure mission using SC1_CFG settings
`make all`	Build all components (FSW + sim + GSW)
`make fsw`	Build flight software only
`make sim`	Build simulation components only
`make gsw`	Build ground software only
`make launch`	Launch all components in Docker
`make stop`	Stop all running components

9.3 Docker Deployment

NOS3 components run as Docker containers connected through a Docker Compose network. Each layer (GSW, FSW, sim) runs in its own container, enabling isolated development, testing, and scaling. The Docker overlay network provides the UDP socket connectivity that the generic radio component requires.

For operational ground station deployment, the Docker architecture maps directly to production patterns. The GSW container can be replaced with a YAMCS deployment on bare metal or in Kubernetes. The FSW container provides the same interface regardless of whether it runs in simulation or on target hardware. The simulation containers are simply removed when transitioning to real hardware.

10. Related Work

NOS3 operates in a landscape of spacecraft simulation platforms. This section compares NOS3 with established alternatives.

10.1 Analytical Graphics STK (Systems Tool Kit)

STK is a commercial, closed-source simulation platform widely used in the aerospace industry for mission analysis, orbit determination, and link budget calculations. STK provides high-fidelity environmental models and extensive visualization but lacks the integrated flight software execution and ground station interface that NOS3 provides. STK simulates the physics; NOS3 runs the actual flight software.

10.2 NASA GMAT (General Mission Analysis Tool)

GMAT is NASA's open-source mission analysis tool for orbit design, maneuver planning, and trajectory optimization. Like STK, GMAT focuses on the orbital dynamics and mission planning domain. It does not include flight software execution, ground station simulation, or hardware component models. GMAT and NOS3 are complementary: GMAT designs the orbit, NOS3 simulates the spacecraft flying it.

10.3 OpenSatKit

OpenSatKit provides an open-source satellite development kit with integrated ground station tools. It offers a narrower scope than NOS3, focusing on single-component development rather than full mission simulation. OpenSatKit lacks the 21-component simulator library, the 42 Dynamics engine, and the multi-GSW support that characterize NOS3.

10.4 NASA CORE (Core Reusable Engineering)

NASA's CORE initiative aims to create reusable engineering artifacts across missions. While CORE focuses on design patterns and shared components at the engineering process level, NOS3 provides an executable simulation environment. CORE defines what should be shared; NOS3 provides the platform where shared components are exercised.

10.5 Comparative Summary

Feature	NOS3	STK	GMAT	OpenSatKit
FSW Execution	Yes (cFS, F Prime)	No	No	Limited

Feature	NOS3	STK	GMAT	OpenSatKit
Ground Software	4 options	Proprietary	No	Basic
Orbital Dynamics	42 (good)	Excellent	Excellent	Basic
Hardware Simulation	21 components	Sensors only	No	Minimal
HITL Support	Yes	Partial	No	No
Open Source	Apache 2.0	No	Apache 2.0	MIT
CCSDS Compliance	Full	Partial	No	Partial
Cost	Free	Commercial	Free	Free

NOS3's distinguishing advantage is its vertical integration: it is the only platform that combines flight software execution, ground station operation, orbital dynamics, and hardware simulation in a single open-source environment. Other platforms excel in individual domains, but NOS3 provides the complete mission stack.

11. Conclusion & Future Work

NASA's NOS3 represents a significant advance in accessible spacecraft simulation. By providing a complete, open-source mission stack—from ground software through flight software to orbital dynamics and hardware models—NOS3 enables organizations to develop, test, and validate spacecraft systems without requiring access to expensive proprietary tools or dedicated test facilities.

The analysis presented in this paper identifies YAMCS as the optimal ground software backend for ground station modernization, owing to its web-native architecture, CCSDS compliance, and REST API integration capabilities. The generic radio's UDP socket interface provides a clean protocol boundary that can be bridged to API-driven ground station services through a translation layer.

The four-phase integration roadmap provides a structured path from simulation to operations. By progressively replacing simulated components with real hardware and infrastructure, the validated software stack transitions from NOS3's Docker containers to production deployment with minimal risk.

Future Work

Several areas merit further investigation:

- **Real hardware bridging:** The protocol bridge between NOS3's UDP interfaces and SDR hardware requires implementation and validation with actual RF equipment. Characterizing the latency, jitter, and error-rate differences between simulated and real radio links will inform operational procedures.
- **Multi-spacecraft simulation:** NOS3 supports multiple spacecraft in its mission configuration. Extending the integration roadmap to constellation operations—with multiple spacecraft communicating through a shared ground station—introduces scheduling, resource allocation, and cross-link coordination challenges.
- **On-orbit AI integration:** The `onair` component provides on-orbit AI inference capabilities. Investigating how edge AI can augment ground station operations—for example, by pre-filtering telemetry on the spacecraft or triggering autonomous responses to detected anomalies—represents a promising research direction.
- **Operational deployment:** Transitioning from a YAMCS deployment in Docker containers to a production Kubernetes environment requires addressing persistence, redundancy, monitoring, and incident response procedures that are beyond the scope of simulation validation.
- **Security hardening:** The cryptolib component provides cryptographic services for flight software. Extending end-to-end security from the spacecraft through the ground station—including authenticated command uplink, encrypted telemetry downlink, and key management—requires careful integration with ground station security infrastructure.

NOS3's open-source license (Apache 2.0) and modular architecture make it an ideal foundation for these investigations. The platform's continued development by NASA's IV&V team, combined with a growing open-source community, positions NOS3 as a sustainable platform for both research and operational ground station development.

---

12. References

1. NASA Independent Verification and Validation Facility, *NOS3: Operational Simulator for Space Systems*, GitHub Repository, 2024. Available: https://github.com/nasa/nos3

2. NASA IV&V, *NOS3 Documentation*, Read the Docs, 2024. Available: https://nos3.readthedocs.io
3. NASA Goddard Space Flight Center, *core Flight System (cFS)*, GitHub Repository, 2024. Available: https://github.com/nasa/cFS
4. NASA Jet Propulsion Laboratory, *F Prime: A Flight-Proven, Open-Source Framework for Embedded Systems*, 2024. Available: https://fprime.jpl.nasa.gov
5. Space Applications Services, *YAMCS: Yet Another Mission Control System*, 2024. Available: https://yamcs.org
6. Ball Aerospace / BAE Systems, *COSMOS: Comprehensive Open-Source Solution for Mission Operations Systems*, 2024. Available: https://cosmosrb.com
7. OpenC3 Inc., *OpenC3: Open Source Command and Control*, 2024. Available: https://openc3.com
8. CCSDS, *CCSDS 727.0-B-5: CCSDS File Delivery Protocol (CFDP)*, Consultative Committee for Space Data Systems, 2020.
9. CCSDS, *CCSDS 133.0-B-2: Space Packet Protocol*, Consultative Committee for Space Data Systems, 2020.
10. E. M. Gertz, *42: A General-Purpose Spacecraft Dynamics Simulator*, NASA Engineering and Safety Center, 2019.
11. Analytical Graphics Inc., *STK: Systems Tool Kit*, AGI/Ansys, 2024. Available: https://www.agi.com/products/stk
12. NASA Goddard Space Flight Center, *GMAT: General Mission Analysis Tool*, 2024. Available: https://gmat.gsfc.nasa.gov
13. T. W. Flatley and D. R. Skillman, *"Kepler's Equation, the Eccentric Anomaly, and the Mean Anomaly,"* NASA Goddard Space Flight Center Technical Report, 2018.
