Dynamic Satellite Simulator (DSS): A Digital Twin Application Case for LEO Mission Validation

Martín Javier Noguerol[1] and Jeremías Pino Demichelis [1]

[1] INVAP SAU, Argentina

Abstract

Traditional flight software validation requires extensive testing on representative hardware, creating schedule bottlenecks by serializing tasks that could otherwise run in parallel. This paper presents the Dynamic Satellite Simulator (DSS), an architectural case study of a digital twin platform developed for Low Earth Orbit (LEO) Earth observation missions. Built on an in-house implementation of the European Space Agency Simulation Model Portability 2 (SMP2) standard, called the Simulation and Ground Software Common Core (SGS-CC), the DSS achieves interoperability between models developed by multiple specialized teams and delivers soft real-time performance. The simulator integrates high-fidelity physical models of spacecraft attitude and orbit, with environmental perturbations and realistic sensor and actuator dynamics, together with functional models of the power, thermal, communications, propulsion and payload subsystems. A flight processor emulator running the unmodified binary of the flight software, a JSON-RPC 2.0 control interface, and a harness service supporting buses and pins across simulated and physical domains, complete the platform. The paper argues that the central architectural contribution is the unification of the emulator and the simulated models under a single SMP2 scheduler, which renders the boundary between simulation and emulation operationally transparent. Field evidence from an in-progress mission includes 81% of Flight Operations Procedures validated virtually before hardware integration, 53,5% of FDIR test cases executed on the DSS, a successful HIL integration over the CAN bus between the simulator and a project EGSE, and the early detection of a watchdog-related flight software defect.

Keywords: satellite simulator, SMP2, software architecture, flight software, virtual validation

Simulador Dinámico de Satélite (DSS): Un Caso de Aplicación de Gemelo Digital en la Validación de Misiones LEO

Resumen

La validación tradicional del software de vuelo exige pruebas exhaustivas sobre hardware representativo, generando cuellos de botella en el cronograma de fabricación. Este trabajo presenta el Simulador Dinámico de Satélite (DSS), un caso de estudio arquitectónico de una plataforma de gemelo digital para misiones de observación terrestre en órbita baja (LEO). Construido sobre una implementación propia del estándar *Simulation Model Portability 2* (SMP2) de la Agencia Espacial Europea, denominada *Simulation and Ground Software Common Core* (SGS-CC), el DSS logra interoperabilidad entre modelos múltiples equipos especializados y ofrece desempeño *soft real-time*. Integra modelos físicos de actitud y órbita con perturbaciones ambientales y dinámica de sensores y actuadores, junto con modelos funcionales de los subsistemas de potencia, térmicos, comunicaciones, propulsión y carga útil. Un emulador de procesador que ejecuta el binario sin modificaciones del software de vuelo,

una interfaz de control JSON-RPC 2.0 y un servicio de *harness* que soporta buses y *pins*, completan la plataforma. El trabajo argumenta que la contribución arquitectónica central es la unificación del emulador y los modelos simulados bajo un único planificador SMP2, lo que vuelve operativamente transparente la frontera entre simulación y emulación. La evidencia de campo de una misión en curso incluye 81% de los procedimientos de vuelo validados virtualmente antes de la integración con hardware, 53,5% de los casos de prueba de FDIR ejecutados sobre el DSS, una integración HIL exitosa sobre el bus CAN entre el simulador y un EGSE del proyecto, y la detección temprana de un defecto de software de vuelo asociado al *watchdog*.
Palabras clave: simulador satelital, SMP2, arquitectura de software, software de vuelo, validación virtual

1. Introducción

El desarrollo de un satélite de órbita baja terrestre (*Low Earth Orbit*, LEO) impone un cronograma de validación en el que la disponibilidad de hardware representativo suele ser el cuello de botella principal: la verificación del software de vuelo (*Flight Software*, FSW), la validación de procedimientos operativos y la depuración de algoritmos de control de actitud requieren tradicionalmente acceso a unidades de ingeniería o a modelos de vuelo del propio satélite, unidades escasas, costosas y sujetas a ventanas de uso compartido que fuerzan una serialización artificial de tareas que en principio podrían ejecutarse en paralelo.

El paradigma de *gemelo digital*, acuñado por Grieves (2014) en el contexto de manufactura, ofrece una salida estructural a este problema. La estrategia, conocida en la industria como *shift left*, consiste en comprimir y paralelizar el cronograma mediante plataformas virtuales que permiten desarrollar y validar software antes de que el hardware físico esté disponible, generando que la curva de recursos de ingeniería se redistribuya anticipando software e integración hacia etapas tempranas del proyecto. Intel ha documentado esta práctica para el desarrollo de procesadores desde 2010 (Greene, 2018), y NASA la ha consolidado en el ámbito espacial a través de la iniciativa JSTAR (NASA, 2025), reportando la validación completa del software de vuelo antes de los ensayos ambientales en una misión interna. Este desplazamiento del esfuerzo de verificación libera al hardware físico para las tareas que únicamente pueden hacerse sobre él.

El *Dynamic Satellite Simulator* (DSS) es una plataforma de simulación para misiones LEO de observación de la Tierra que materializa este paradigma en un contexto industrial concreto. Construido sobre una implementación propia del estándar *Simulation Model Portability 2* (SMP2) de la Agencia Espacial Europea (European Space Agency, 2005), el DSS integra modelos de alta fidelidad del subsistema de control de actitud y órbita (*Attitude and Orbit Control Subsystem*, AOCS) con perturbaciones ambientales y dinámica realista de sensores y actuadores, modelos de los subsistemas térmico, de potencia, propulsión, comunicaciones y carga útil, y un emulador del procesador de vuelo que ejecuta el FSW sin modificaciones (Terma, 2024). Un servidor JSON-RPC 2.0 sobre TCP/IP expone el control y la inspección de la simulación.

Contribuciones. Este trabajo presenta cuatro contribuciones arquitectónicas: (i) una implementación propia del estándar SMP2 —el *Simulation and Ground Software Common Core* (SGS-CC)— que desacopla el núcleo de simulación de los modelos, distribuidos como bibliotecas dinámicas independientes, y habilita el desarrollo paralelo por equipos especializados; (ii) un esquema de intercomunicación entre modelos *pines/harness* (señales físicas o eléctricas) y *buses/nodes* (protocolos de comunicación espaciales: SpaceWire, MIL-STD-1553, CAN, RS-422) extendiendo SMP2; (iii) una única superficie de control de la simulación vía JSON-RPC; (iv) la

integración de un emulador de procesador de vuelo que preserva la representatividad binaria del FSW sobre la misma plataforma.
El argumento central del paper es que la combinación de un núcleo SMP2 con un emulador comercial de procesador, unificados bajo un único *scheduler*, habilita una estrategia de *shift left* demostrable por los datos de campo: el simulador absorbe trabajo de verificación que tradicionalmente requería hardware, y lo libera para aquello que sólo el hardware puede validar.

2. Conceptos preliminares y trabajo relacionado

SMP2 fue introducido por la Agencia Espacial Europea como estándar de portabilidad de modelos de simulación entre misiones y organizaciones (European Space Agency, 2005), con un metamodelo de componentes y un *mapping* completo a C++. El estándar prescribe una arquitectura de componentes y servicios bien tipados, y deja la lógica de cada modelo a la implementación en C++ del desarrollador buscando facilitar el reuso entre proyectos y organizaciones.
El entorno de simulación de SMP2 provee un conjunto de servicios estandarizados: son obligatorios el Logger, el Scheduler —que invoca los puntos de entrada de los modelos ante eventos cíclicos o temporizados—, el Time Keeper —que administra el tiempo de simulación y lo relaciona con los tiempos de época, de misión y Zulu— y el Event Manager, que ofrece un mecanismo global de notificación; dejando los servicios Link Registry y el Resolver como opcionales, y el estándar admite además servicios definidos por el usuario. El metamodelo se especifica en un lenguaje de definición basado en XML (SMDL) y se mapea a C++, y cada modelo atraviesa un ciclo de vida explícito —publicación de campos y operaciones, configuración, conexión e inicialización— antes de su ejecución. Esta separación entre la especificación independiente de la plataforma y su implementación es lo que habilita el reuso de modelos entre entornos. El estándar, originado en la ESA, fue posteriormente adoptado por el ECSS como ECSS-E-ST-40-07C (ECSS, 2020).
Aunque parte de la literatura reserva el término digital twin para representaciones virtuales con sincronización automática y bidireccional con un activo físico operativo, en el dominio espacial el término también se utiliza para designar entornos de simulación y emulación de alta fidelidad orientados a la validación de software de vuelo antes de la disponibilidad del hardware o de la operación en vuelo. En particular, la iniciativa JSTAR de NASA emplea la noción de software digital twin para plataformas all-software donde la computadora de vuelo es emulada, los sensores y actuadores son simulados, los binarios del software de vuelo se ejecutan sin modificaciones y el software de operaciones se integra tal como será utilizado en misión. Este trabajo adopta ese sentido del término: el DSS es un software digital twin orientado a validación pre-vuelo, no todavía un gemelo digital operacional con realimentación automática desde un satélite en órbita.
En cuanto al formalismo de modelado, el DSS sigue un esquema de simulación dirigido por un planificador con paso temporal y una cola de eventos, heredado de SMP2, en lugar de un formalismo de eventos discretos puro. El formalismo DEVS (Discrete Event System Specification) (Zeigler et al., 2000), junto con extensiones como DESS/DEVS para sistemas combinados continuos y de eventos discretos, constituye una alternativa formal reconocida para describir sistemas híbridos de esta clase, y en la literatura se han propuesto incluso transformaciones entre representaciones DEVS y SMP2. La adopción de SMP2 en este trabajo responde a la portabilidad de modelos y a la interoperabilidad con el ecosistema de simulación espacial, más que a la preferencia por un formalismo de modelado en particular.
Varios simuladores satelitales de referencia comparten problemas y soluciones con el DSS. SIMSAT, desarrollado por el *European Space Operations Centre* e integrado en el *EGOS Modelling Framework*, implementa SMP2 sobre una infraestructura que cubre el ciclo de vida completo (Sebastiao et al., 2010; Ellsiepen et al., 2014). SimTG de

Airbus Defence and Space y BASILES de CNES son implementaciones industriales adicionales del estándar (Mollier, 2015).

3. Requisitos arquitectónicos

El diseño del DSS respondió a tres requisitos centrales basados en la experiencia de simuladores previos dentro de la organización:

R1 — Integración modular de modelos desarrollados por equipos especializados. Permitir que modelos críticos producidos fuera del equipo del simulador —en particular los modelos del subsistema de control de actitud y órbita (AOCS), desarrollados por los especialistas de guiado, navegación y control— se integren al simulador como componentes independientes, con sus propios ciclos de versión, sin requerir cambios estructurales en el resto del modelo del satélite.

R2 — Representatividad del software de vuelo. Ejecutar el FSW sin modificaciones —la imagen binaria exacta del vuelo—, lo que obliga a emular procesadores *rad-hard* (típicamente SPARCv8).

R3 — Soft real-time con tolerancia controlada. Avanzar la simulación en tiempo real (de reloj de pared), con desviaciones acotadas sin perder determinismo.

4. Arquitectura general del DSS

El DSS se organiza en tres componentes principales: el núcleo de simulación SGS-CC, un conjunto de bibliotecas dinámicas que contienen los modelos de los subsistemas del satélite, y una interfaz gráfica de usuario (GUI). La Figura 1 presenta la vista estática de alto nivel de esta arquitectura. El acoplamiento entre los tres componentes es explícito y mínimo: el SGS-CC expone un servidor JSON-RPC sobre TCP/IP como única superficie de control de la simulación, y las bibliotecas de modelos se cargan dinámicamente.

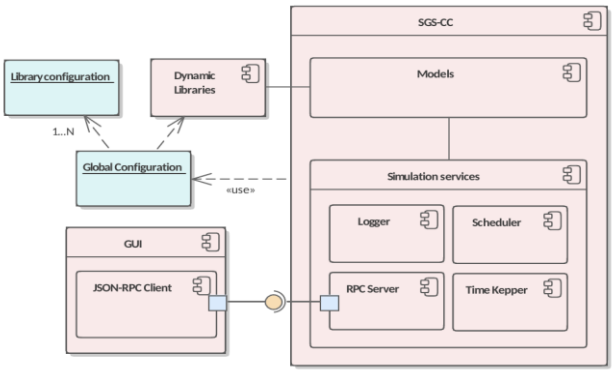

Figura 1. *Diagrama UML de componentes. Vista estática de alto nivel de la arquitectura del DSS. El SGS-CC encapsula los servicios de simulación (Scheduler, Time Keeper, RPC Server, Logger) y el contenedor de Modelos, que se cargan desde Dynamic Libraries externas configuradas mediante un archivo Global Configuration. La GUI accede al SGS-CC a través de un cliente JSON-RPC.*

Notación UML: los recuadros con ícono de componente representan componentes; el círculo indica una interfaz provista; el semicírculo, una interfaz requerida; y la flecha punteada, una relación de dependencia.

Los modelos del satélite se organizan en dos categorías: modelos físicos y modelos de subsistema. Los primeros calculan magnitudes derivadas de leyes de la física sin correspondencia directa con hardware (propagación de actitud con cuerpos flexibles,

propagación orbital con métodos numéricos específicos, efemérides de Sol, Luna y Tierra, colección de perturbaciones externas). Los métodos numéricos que gobiernan la evolución de estos modelos físicos son métodos establecidos: la dinámica de actitud se integra mediante LSODE y la órbita se propaga con SGP4. Estos solvers son implementados por los especialistas de AOCS y el DSS los integra en su infraestructura de simulación. Los segundos representan unidades o grupos de hardware real y simulan su comportamiento: plataforma de servicio (AOCS, PCS, TCS, Propulsión, OBC) — donde PCS denota el subsistema de control de potencia (Power Control Subsystem), TCS el subsistema de control térmico (Thermal Control Subsystem) y OBC la computadora de a bordo (On-Board Computer)— y carga útil (instrumentos ópticos y de colección, transpondedor en banda S, transmisor en banda X, caja de ingesta de datos). Vale observar que el alcance del TCS en la configuración aquí presentada cubre la lógica de actuación y telemetría térmica (heaters, termistores, termostatos, zonas térmicas) pero no resuelve el estado térmico dinámicamente; la plataforma dispone de un motor de cálculo térmico completo, utilizado en otras configuraciones de misión, cuya integración en esta configuración no fue solicitada.

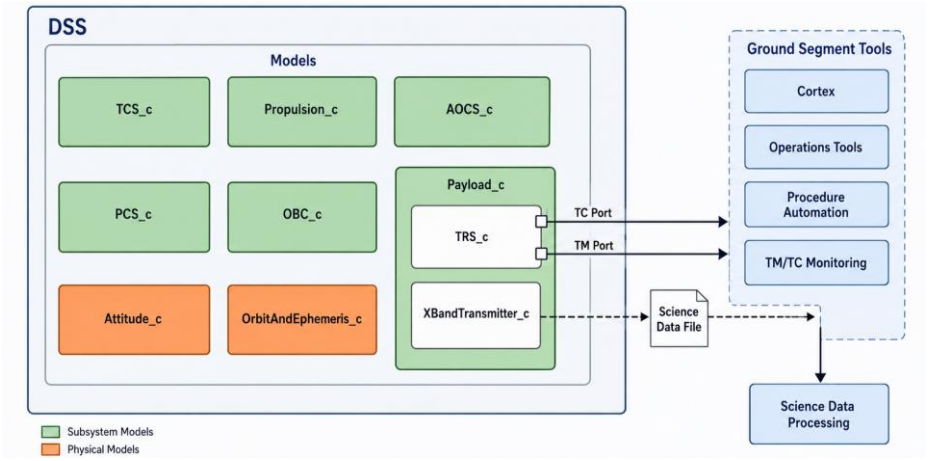

Figura 2. *Descomposición de modelos del DSS. En tonos diferenciados aparecen los modelos de subsistema (TCS, Propulsión, AOCS, PCS, OBC, Payload) y los modelos físicos (Attitude, Orbit and Ephemeris). El contenedor Payload expone los puertos de telecomando (TC Port) y telemetría (TM Port) al entorno externo mediante el submodelo TRS, y el submodelo XBandTransmitter produce el archivo de datos científicos virtuales (sintéticos) que el satélite recolectaría de la superficie terrestre durante la simulación.*

El patrón arquitectónico que emerge de esta separación es el siguiente: los modelos físicos publican, periódicamente, sus soluciones (cuaternión de actitud, posición y velocidad, flujo solar, campo magnético local) a los modelos de subsistema que las consumen como fuentes de información para sus sensores y actuadores. Inversamente, los modelos de subsistema que afectan al estado físico del satélite (propulsores, ruedas de reacción, barras de torque magnético) lo hacen a través de interfaces C++ estandarizadas que el colector de perturbaciones de los modelos físicos recorre para integrar sus aportes. Esta separación estricta entre la física del vehículo y la implementación de las unidades permite reemplazar un modelo de sensor por otro sin tocar el modelo de actitud.

5. Núcleo de simulación SGS-CC

El SGS-CC (*Simulation and Ground Software Common Core*) es el núcleo de ejecución del DSS.

5.1 Servicios de simulación

Siguiendo el estándar SMP2, el SGS-CC implementa los servicios obligatorios: gestor de tiempo (*Time Keeper*), planificador de eventos (*Scheduler*), logger, y registro de modelos instanciados en un árbol jerárquico navegable. El *Time Keeper* mantiene cuatro referencias temporales simultáneas accesibles por el usuario: *simulation time* (tiempo interno de simulación, iniciado en cero), *mission time* (tiempo desde el evento de inicio de misión), *epoch time* (referenciado al epoch J2000) y *Zulu time* (UTC).
El *Scheduler* implementa una cola de eventos ordenada por tiempo futuro, con soporte para eventos recurrentes parametrizados por periodo y cantidad de repeticiones. Los eventos pueden provenir de la API JSON-RPC (comandos, inyecciones de falla y cambios de parámetros programados) o del propio avance de los modelos (muestreo periódico de sensores, actualización de modelos físicos). Entre los servicios *Scheduler* y el *Time Keeper* cooperan para calibrar las pausas necesarias entre pasos de simulación y mantener la relación configurada entre tiempo simulado y tiempo de ejecución. Esa relación es un factor de velocidad configurable: el caso nominal es 1:1 (tiempo real, de reloj de pared), pero puede acelerarse para campañas prolongadas o ralentizarse para depuración. El avance procede en pasos discretos de tamaño configurable y, en cada paso, el planificador despacha los eventos vencidos de su cola, los modelos de tiempo discreto actualizan su estado y los modelos físicos integran sus ecuaciones diferenciales sobre el intervalo del paso; de este modo conviven sobre una misma base temporal la evolución continua de la física, las actualizaciones de tiempo discreto y los eventos asincrónicos encolados.
Además de los servicios SMP2 obligatorios, el SGS-CC incorpora tres servicios específicos del proyecto: un servicio de *harness* (descrito en la Sección 6) para conectar interfaces entre modelos, un servicio de inyección de fallas, y un servicio de *store/restore* capaz de serializar el estado completo de la simulación —incluido el estado del emulador de procesador— a un archivo persistente y restaurarlo posteriormente.

5.2 Carga dinámica de modelos y servicios

En el SGS-CC los modelos y los servicios adicionales no se compilan dentro del núcleo sino que se cargan dinámicamente desde bibliotecas compartidas Linux (.so). El núcleo es agnóstico respecto de qué modelos se instanciarán; un archivo YAML de configuración global, leído al arranque, enumera los modelos a cargar, sus clases, sus librerías y sus archivos de configuración específicos.

5.3 API de control e inspección

La interacción con el SGS-CC para controlar la simulación se canaliza íntegramente a través de un servidor JSON-RPC 2.0 (JSON-RPC Working Group, 2013) que el núcleo levanta al arrancar en un puerto TCP configurable. La API se organiza en tres familias de métodos. La primera, *control de simulación*, expone arranque, pausa, abortar, consulta de estados SMP2, control del factor de velocidad y persistencia. La segunda, *inspección y manipulación de modelos*, expone la navegación del árbol SMP2, la lectura de variables con tipos primitivos y compuestos, la escritura puntual de valores (`set`) para configurar condiciones iniciales o ajustar parámetros en tiempo de ejecución, y la escritura sostenida (`override`) que fuerza el valor visible de una variable independientemente de lo que el modelo calcule internamente, hasta que el override se libera. La tercera, *inyección de fallas*, expone activación y desactivación parametrizadas por ruta SMP2 y nombre de falla. Las dos últimas familias —`override` e inyección de fallas— son los mecanismos centrales de construcción de

escenarios de prueba que requieren forzar al sistema a comportamientos específicos sin modificar la lógica interna de los modelos.
Muchos de los métodos son *schedulable*: aceptan argumentos opcionales que permiten diferir su ejecución a un momento futuro de la simulación. Esta capacidad habilita la construcción de procedimientos de vuelo automatizados en Python o *Robot Framework* (un framework de automatización de pruebas guiado por palabras clave; Robot Framework Foundation, 2024): el procedimiento programa una secuencia completa de acciones contra el simulador como si fuera una campaña real. La biblioteca cliente en Python, distribuida como parte del paquete, envuelve cada método RPC en una función con firma pythónica. La propia GUI es una aplicación en Python que se comunica con el SGS-CC exclusivamente a través de esta biblioteca cliente JSON-RPC.

6. Modelo de componentes y comunicación interna

Una vez instanciados, los modelos del satélite deben comunicarse entre sí para representar fielmente la dinámica real. SMP2 provee dos mecanismos genéricos —eventos y puertos tipados— pero, en el contexto de un simulador, ninguno de los dos captura directamente dos abstracciones específicas del dominio: las interfaces físicas (aplicación de torques, consumo eléctrico) y los protocolos de comunicación espacial (SpaceWire, MIL-STD-1553, CAN, UART, RS-422). El SGS-CC extiende prácticamente SMP2 con dos mecanismos complementarios: interfaces C++ de fenómeno físico, y un servicio de *harness* que conecta modelos mediante *pins* y *buses/nodes*.

6.1 Interfaces C++ de fenómeno físico

Los fenómenos físicos del vehículo (torques, fuerzas perturbadoras, consumo eléctrico) se resuelven mediante interfaces C++ base que todo modelo que produzca o consuma el fenómeno debe implementar. Tres interfaces son especialmente relevantes: `IRWTorque`, implementada por las ruedas de reacción (métodos `GetTorque()` y `GetAngularMomentum()`, que el modelo de actitud suma al recorrer el árbol); `IForcePerturbations`, implementada por propulsores, barras de torque magnético y otros efectores que introducen perturbaciones externas (`GetForce()` y `GetTorque()`); y `Load_c`, clase base de todo componente eléctricamente activo (`GetPowerConsumption()` que el modelo de potencia integra al actualizar el voltaje del bus, y que en configuraciones con motor térmico completo se extiende para reportar disipación).
La mecánica de consulta es un patrón de visitor sobre el árbol SMP2, expuesto como extensión del servicio del simulador. Los productores del fenómeno no conocen al consumidor y viceversa; lo único compartido es la interfaz.

6.2 Harness: pins y buses/nodes

Los fenómenos eléctricos y de señal a nivel de cableado —alimentaciones, tierras, telemetrías analógicas y digitales, comandos discretos, buses de comunicación espaciales— requieren una abstracción más rica que una interfaz C++. Para estos casos el SGS-CC provee un servicio de *harness* inspirado en la arquitectura eléctrica real del satélite.
El *harness* define dos tipos de conexiones. Los *pins* son conexiones punto a punto que propagan información de una o más fuentes a uno o más sumideros, con cuatro variantes (`InputPin`, `OutputPin`, `BiPin`, `MultiInputPin`); se utilizan para alimentaciones, tierras, telemetrías analógicas, estados digitales y para propagar magnitudes físicas derivadas hacia los sensores (cuaternión de actitud, posición orbital, campo magnético local). Los *buses/nodes* modelan protocolos de comunicación espacial: el *bus* representa el medio físico compartido, el *node* encapsula el comportamiento de envío y recepción del protocolo en un extremo. El esquema soporta direccionamiento específico del protocolo, formatos de mensaje propios, y la conexión de *monitors* para

logging y depuración. Los *buses* exponen además métodos de inyección directa de mensajes y logueo desde la API JSON-RPC. El modelado abstrae el medio compartido y se concentra en las capas de protocolo por encima del nivel físico —direccionamiento, formato de mensajes y semántica de envío y recepción—; el nivel físico de señalización no se representa, por tratarse de una simulación. El conjunto preciso de capas modeladas depende del protocolo (por ejemplo, MIL-STD-1553, RMAP o CANopen). Un caso de uso relevante del mismo mecanismo es la conexión entre un *bus* simulado y su contraparte física real: el servicio de *harness* admite un *node* especial cuya implementación traduce mensajes hacia y desde una interfaz de hardware externa. El resultado es un puente transparente que habilita configuraciones HIL a nivel de bus, en los que un modelo de equipo se reemplaza por su hardware real sin tocar el resto de la simulación.

6.3 El patrón general

Combinando ambos mecanismos emerge el patrón que gobierna toda la intercomunicación dentro del DSS: los modelos físicos publican sus soluciones hacia los modelos de subsistema a través de *pins*, los modelos de subsistema procesan comandos que les llegan desde el FSW por buses de comunicación, producen telemetrías y aportan sus efectos al estado físico del vehículo a través de las interfaces C++. El ciclo se cierra cada paso de simulación. Este patrón permite intercambiar la implementación de un sensor o actuador por otra de distinta fidelidad sin tocar ni el modelo físico ni el resto del subsistema, de esta manera el FSW ejecutándose dentro del emulador se comunica con el resto del satélite únicamente a través de *buses*, exactamente como lo hace en el hardware real. En el plano de la implementación, los publicadores son los modelos físicos —que exponen sus magnitudes a través de pins— y los suscriptores son los modelos de subsistema que las consumen (por ejemplo, un sensor que lee la actitud propagada). El enlace entre publicadores y suscriptores se resuelve recorriendo el árbol jerárquico de modelos SMP2 con un patrón visitor: al inicializar la simulación, un visitante recorre el árbol, descubre las interfaces provistas y requeridas de cada modelo y establece las conexiones C++ correspondientes, que luego se evalúan con la periodicidad de cada paso de simulación.

7. Interfaces externas y control remoto

Más allá del control de la simulación vía JSON-RPC descrito en la Sección 5, el DSS expone un segundo conjunto de interfaces hacia el exterior: las interfaces *satelitales* propiamente dichas, que son las mismas que tendría el satélite real frente al segmento terreno. La separación conceptual entre ambas categorías es importante. La primera (JSON-RPC) es una interfaz de *instrumentación* del simulador, no existe en el vehículo físico, y sirve para controlar la simulación. La segunda (TC, TM, datos científicos) es una interfaz de *misión*: reproduce la electrónica de vuelo, opera con los mismos formatos que se transmitirán por radiofrecuencia, y es consumida por herramientas del segmento terreno que, en principio, no necesitan saber si están hablando con un satélite real o con un simulador.

7.1 Puertos de telecomando y telemetría

El modelo de transpondedor en banda S (TRS) es el responsable de exponer hacia el exterior los puertos de telecomando y telemetría del vehículo. El simulador instancia dos modelos TRS redundantes —reflejando la redundancia del hardware real— cada uno compuesto por un transmisor y un receptor, lo que resulta en dos puertos TCP/IP para telecomandos y dos para telemetría. Los puertos son deliberadamente TCP crudo, sin protocolo de aplicación agregado: los bytes que entran por el puerto de telecomando son los mismos que entrarían al decodificador de TC del hardware real, y los bytes que salen por el puerto de telemetría son los frames CCSDS (Consultative Committee for Space Data Systems) que emitiría el formateador de TM real.

El receptor de TC, cuando está encendido y operacional, entrega el mensaje al modelo SUM (*Service Utility Module*) de la computadora de vuelo, el cual efectúa verificaciones previas a la aceptación. El transmisor de TM, por su parte, sólo emite datos cuando el modelo se encuentra encendido y en estado de modulación, replicando el comportamiento del transpondedor real.

7.2 Datos científicos

La descarga de datos científicos, simulada por el modelo de *X-Band Transmitter*, produce un flujo de datos científicos que en vuelo se transmitiría por el enlace descendente de banda X. La interfaz en el DSS, simplificada, es un archivo con el formato esperado por las herramientas de procesamiento del operador, escrito en una ubicación configurable; el contenido es, en esta configuración, un payload aleatorio compatible con el formato real, suficiente para validar el pipeline de ingestión del segmento terreno.

7.3 Integración con herramientas del segmento terreno

Todas las interfaces externas del DSS usan protocolos estándar como TCP/IP y sistema de archivos. Simulador, herramientas de operación y operador humano pueden residir en la misma máquina durante el desarrollo o distribuirse durante campañas formales de validación, sin cambios de configuración más allá de direcciones IP y puertos. Esta propiedad es el fundamento del uso del DSS como digital twin funcionalmente intercambiable con el satélite real. En la configuración actual los puertos TC/TM del modelo TRS se conectan al simulador de la herramienta Cortex de SAFRAN.

8. Integración con el software de vuelo

La representatividad del software de vuelo en el DSS se logra mediante emulación del procesador de vuelo, no mediante simulación funcional de sus efectos. La distinción es consecuencia directa del requisito R2: el binario exacto que se cargará en el hardware volador es el que se ejecuta dentro del simulador, sin recompilación ni portado a arquitectura del host. El emulador utilizado es TEMU, de la firma alemana Terma. TEMU se distribuye bajo licencia comercial. Para procesadores SPARC/LEON existen además simuladores a nivel de instrucción como TSIM (Gaisler, 2024); el DSS emplea TEMU por su capacidad de ejecutar sin modificaciones el binario de vuelo dentro del mismo planificador que el resto de los modelos.

El emulador se integra al DSS como parte del modelo de OBC, específicamente como modelo interno al *Processor Module* (PM). El FSW se sincroniza periódicamente con el PPS, y esta sincronización se reproduce fielmente en el simulador.

La decisión arquitectónica con mayor implicación operativa es que el emulador y el resto de los modelos comparten el mismo planificador SMP2 del SGS-CC. Esto es, en rigor, el beneficio central de adoptar SMP2 como marco del simulador: el núcleo trata al emulador como a cualquier otro submodelo, y el planificador avanza sus ciclos en la misma línea de tiempo virtual en la que avanza los demás. No hay dos relojes que haya que alinear, ni hilos independientes que necesiten negociación, ni ventanas de tolerancia entre el avance del FSW y el de los modelos físicos. Los mensajes entre FSW y modelos de subsistema viajan por las interfaces internas del OBC —SpaceWire, MIL-STD-1553 y otras, modeladas con el *harness* descrito en la Sección 6.2— exactamente como en el hardware real. El FSW no percibe diferencia operativa entre correr en el emulador o en el hardware.

9. Casos de aplicación y evaluación

El DSS se aplica actualmente en una misión LEO de observación terrestre en desarrollo. Este apartado reporta el uso del simulador en cinco escenarios de aplicación, con evidencia cualitativa y cuantitativa de campañas reales del proyecto.

9.1 Reuso de modelos desarrollados para el diseño de algoritmos de control

Los modelos físicos (actitud con cuerpos flexibles, órbita y efemérides con perturbaciones) de sensores y actuadores utilizados para iterar sobre el desarrollo de la lógica de control son embebidos en el simulador. Este entorno, con el FSW, se utilizó posteriormente para verificar los algoritmos ya implementados en el software de vuelo.

9.2 Verificación de procedimientos de vuelo

El uso más intensivo del DSS hasta la fecha corresponde a la verificación de procedimientos de operación de vuelo (*Flight Operations Procedures*, FOPs). El 81% de los procedimientos fueron desarrollados en el ambiente virtual antes de existir el hardware para su prueba. La ejecución se realiza desde las herramientas nominales de control del satélite, conectadas al DSS mediante el modelo de Cortex descrito en la Sección 7.3, con lo cual el procedimiento ejecutado en simulación es literalmente el mismo script que se utilizará durante la operación real.

Concretamente, el simulador ayudó a identificar inconsistencias en la base de datos compartida entre segmento terreno y segmento de vuelo, tanto en telecomandos como en telemetrías: valores de parámetros fuera de rango, definiciones inconsistentes entre ambos lados de la cadena, discrepancias de tipos, etc. Esta detección temprana representa un ahorro neto de horas de integración.

Un caso particular merece mención: durante la validación de ciertos procedimientos que incluían la solicitud del reporte de pasos de macrocomandos, se observó que el FSW se reseteaba por detención del watchdog. La raíz del problema era una tarea del FSW cuyo tiempo de ejecución excedía, bajo ciertas condiciones, la tolerancia del watchdog configurada. La reproducibilidad del fenómeno en simulación, permitió diagnosticar la causa y corregir el problema mediante una actualización de la tolerancia del watchdog para esa tarea específica. Detectar este defecto durante integración con hardware, habría representado un costo sensiblemente superior.

9.3 Validación cruzada con ensayos funcionales de propulsión

El desarrollo de los procedimientos de vuelo para maniobras de propulsión se realizó sobre el DSS, y los mismos procedimientos se usaron posteriormente en los ensayos funcionales del subsistema con hardware real.

En esta aplicación el DSS contribuyó a estabilizar el entendimiento operativo antes de tocar hardware: la secuencia de pasos necesaria para subir una maniobra, los tiempos que demanda cada paso, las ventanas de validación y las condiciones de aceptación del FSW se consolidaron primero en simulación. Cuando el ensayo funcional se ejecutó sobre el hardware, el procedimiento ya había sido depurado, y los incidentes observados fueron únicamente los atribuibles al hardware físico.

9.4 Validación de FSW: campañas de pruebas y filtrado pre-hardware

Un caso destacable del valor del DSS como entorno de validación de FSW es la campaña de pruebas de las reglas de detección, aislamiento y recuperación de fallas (Fault Detection, Isolation and Recovery, FDIR). La estrategia de validación distribuyó deliberadamente los casos de prueba entre tres entornos buscando una cobertura representativa: 53,5% se validaron sobre el DSS, 42,7% sobre el modelo de ingeniería del satélite y 3,8% sobre el modelo de vuelo. Cada caso fue asignado al entorno de mínima fidelidad capaz de producir un resultado concluyente sobre la regla bajo prueba, y los casos típicamente incluyen tanto la cadena nominal como la redundante para verificar el comportamiento de las redundancias además de la regla misma.

Más allá de la campaña FDIR, el DSS sostiene la validación regular del FSW como uno de los entornos de prueba de cada nueva versión liberada del software de vuelo, junto con el modelo de ingeniería y el modelo de vuelo. Cada release del FSW se integra manualmente en los distintos setups y se ejecutan las suites que corresponden al contexto de validación deseado. La política del cliente exige que el conjunto completo de tests del FSW se valide finalmente sobre hardware, pero la ejecución previa sobre el DSS funciona como filtro económico: una corrida completa en hardware insume más

de dos días de uso del modelo de ingeniería, por lo que detectar y corregir defectos en simulación antes de la transferencia ahorra tiempo de hardware crítico para el cronograma del proyecto.
En cobertura, sobre el total de casos de prueba definidos para el FSW, 72% se ejecutan rutinariamente sobre el DSS antes de su corrida sobre el modelo de ingeniería, produciendo resultados equivalentes en ambos entornos; los restantes están reservados al hardware por diseño de validación.

9.5 Integración Hardware-in-the-Loop sobre bus CAN

La capacidad HIL a nivel de bus descrita en la Sección 6.2 se validó en una prueba de concepto que conectó un bus CAN simulado dentro del DSS con un bus CAN físico externo, este último vinculado a un EGSE (Electrical Ground Support Equipment) del proyecto. El puente se realizó a través de un node HIL implementado al efecto en el servicio de harness, mediado físicamente por un adaptador comercial CAN-USB. La caracterización temporal del puente, realizada mediante mediciones de loopback con marcas de tiempo en ambos extremos y verificación con osciloscopio, mostró que la representación es fiel para tamaños de paso de simulación iguales o mayores a 5 ms. Esta resolución temporal cubre las necesidades operativas de la mayor parte de las campañas previstas; las campañas que requieran tamaños de paso menores constituyen un caso identificado de trabajo futuro que demandará un rediseño del puente.
Más allá del aspecto puramente técnico, el resultado relevante es que el DSS quedó demostrado como pieza integrable en la cadena de validación física del proyecto: una unidad de hardware o un EGSE conectado al puesto de ensayo se ve, desde la óptica del resto del modelo, como un componente más del simulador, sin requerir adaptaciones específicas de los modelos circundantes. Esta capacidad complementa el modo enteramente virtual del DSS y posiciona al simulador como soporte de la fase de integración del satélite, además de la fase puramente de desarrollo.

10. Discusión, lecciones aprendidas y limitaciones

La unificación de la línea temporal entre emulador y modelos simulados bajo un único planificador pagó dividendos que superaron la expectativa inicial. Incluso por sobre la portabilidad teórica de modelos entre cores de simulación —difícil de ejercer en la práctica por los servicios optativos específicos de cada implementación—. Complementariamente, cargar modelos y servicios como bibliotecas dinámicas en lugar de compilarlos dentro del núcleo demostró ser la habilitación técnica para integrar modelos desarrollados por equipos especializados sin acoplar sus ciclos de release.
Existen limitaciones conocidas. El puente HIL sobre bus CAN, validado en la Sección 9.5, es fiel para tamaños de paso de simulación iguales o mayores a 5 ms; campañas que requieran resolución temporal menor demandarán un rediseño del puente. No se dispone aún de mediciones sistemáticas del desempeño temporal del simulador —factores de velocidad sostenibles, jitter bajo carga— que permitan caracterizar cuantitativamente el margen real del contrato *soft real-time*.
Una observación transversal a las limitaciones anteriores es que la cobertura total del comportamiento satelital por el DSS es inherentemente asintótica. Un simulador de alta fidelidad no es el sustituto del hardware de vuelo sino un complemento que desplaza el esfuerzo hacia etapas tempranas y libera al hardware para aquello que sólo el hardware puede validar.

11. Conclusiones

Este trabajo presentó el Simulador Dinámico de Satélite (DSS), una plataforma de gemelo digital para misiones LEO de observación terrestre, organizada en torno a cuatro contribuciones arquitectónicas: una implementación propia del estándar SMP2 (el SGS-CC) con carga dinámica de modelos, un servicio de harness que extiende SMP2 con pins y buses/nodes, una única superficie de control JSON-RPC que atiende

indistintamente a múltiples clientes, y la integración de un emulador de procesador que ejecuta sin modificaciones el binario del software de vuelo. Por sobre estas contribuciones tomadas aisladamente, el argumento unificador del paper es que el emulador y los modelos simulados comparten un único planificador SMP2: esa unificación es la que vuelve operativamente transparente la frontera entre simulación y emulación, y —a juicio de los autores— el motivo más fuerte para adoptar SMP2 como marco en nuevos proyectos.

La evidencia de campo respalda el enfoque. En cobertura: 81% procedimientos de vuelo validados virtualmente antes de la integración con hardware, 72% casos de prueba del FSW ejecutados rutinariamente sobre el DSS como filtro pre-hardware, y 53,5% casos de prueba FDIR ejecutados sobre el DSS dentro de una estrategia de validación distribuida en tres niveles. En aplicación: integración HIL exitosa sobre bus CAN entre el simulador y un EGSE del proyecto, reuso cruzado entre validación virtual y ensayos funcionales de propulsión, y detección temprana de defectos del FSW que en un esquema tradicional sólo se habrían observado durante integración final.

Varias líneas de trabajo futuro resultan naturales a partir de los resultados presentados. La incorporación al DSS del motor térmico completo ya disponible en otras configuraciones de la plataforma cerrará el modelado físico del vehículo. La caracterización sistemática del desempeño temporal del planificador a través de modos estadísticos permitirá cuantificar el margen real del contrato soft real-time. El rediseño del puente HIL para resolución temporal menor a 5 ms ampliará el alcance de las campañas mixtas. La integración del SGS-CC con emuladores libres como QEMU, manteniendo el mismo principio arquitectónico de unificación bajo el planificador SMP2 discutido en la Sección 8, es una alternativa de interés frente al emulador comercial actual. La extensión del enfoque a la simulación de constelaciones de satélites permitiría validar maniobras de vuelo en formación y otras operaciones cooperativas características de esas arquitecturas. Finalmente, la aplicación del SGS-CC y de los modelos reusables a una segunda misión es el mecanismo natural para validar empíricamente la propiedad de portabilidad entre misiones que el diseño contempló desde el inicio.

Agradecimientos

Los autores agradecen al equipo de desarrollo de software de vuelo, operaciones y Leonardo Handsztok (CTO GIP) por sus aportes a este trabajo.

Durante la preparación de este trabajo, el/los autor(es) utilizaron Claude con el fin de mejorar el lenguaje y la legibilidad del manuscrito. Tras utilizar esta herramienta/servicio, el/los autor(es) revisaron y editaron el contenido según fuera necesario y asumen plena responsabilidad por el contenido de la publicación.

Referencias

Allard, C., Ramos, M. D., Schaub, H., Kenneally, P. y Piggott, S. (2018). Modular software architecture for fully coupled spacecraft simulations. *Journal of Aerospace Information Systems*, 15(12), 670-683. https://doi.org/10.2514/1.I010653

ECSS. (2020). *Space engineering — Simulation modelling platform* (ECSS-E-ST-40-07C). European Cooperation for Space Standardization.

Ellsiepen, P., Fritzen, P. y Reggestad, V. (2014). ESA's model based approach for the development of operational spacecraft simulators. En *SpaceOps 2014 Conference* (AIAA 2014-1866). American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2014-1866

European Space Agency. (2005). *SMP 2.0 handbook* (EGOS-SIM-GEN-TN-0099, Issue 1 Revision 2).

Gaisler. (2024). *TSIM3-LEON3: Instruction-level simulator for LEON2/3/4-based systems*. Cobham Gaisler. https://www.gaisler.com/index.php/products/simulators/tsim3/tsim3-leon3

Greene, M. (2018). Shifting left—Building systems & software before hardware lands. Intel Developer Zone. https://www.intel.com/content/www/us/en/developer/articles/technical/shifting-left-building-systems-software-before-hardware-lands.html

Grieves, M. (2014). *Digital twin: Manufacturing excellence through virtual factory replication* [White paper]. University of Michigan.

JSON-RPC Working Group. (2013). *JSON-RPC 2.0 specification*. https://www.jsonrpc.org/specification

Mollier, J.-J. (2015). SMP2 modelling using the K2 simulation infrastructure. En *Workshop on Simulation for European Space Programmes (SESP)*. European Space Agency.

NASA. (2025). *JSTAR digital twins: Value proposition*. National Aeronautics and Space Administration. https://www.nasa.gov/jstar-digital-twins/

Robot Framework Foundation. (2024). *Robot Framework* [software]. https://robotframework.org/

Sebastiao, N., Segneri, D. y Lyngvi, A. (2010). A new standard for simulation models' portability and its implementation at ESOC. En *SpaceOps 2010 Conference* (AIAA 2010-2268). American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2010-2268

Terma. (2024). *TEMU: Spacecraft flight processor emulator*. Terma A/S. https://www.terma.com/products/space/emulators/

Zeigler, B. P., Praehofer, H. y Kim, T. G. (2000). *Theory of Modeling and Simulation* (2.ª ed.). Academic Press.