IAC-07-D1.3.08

System Engineering Strategy for Distributed Multi-Purpose Simulation Architectures

Dilipkumar Bhula

United Space Alliance, LLC, USA
Dilip.Bhula@usa-spaceops.com

Cindy Marie Kurt

United Space Alliance, LLC, USA
Cindy.M.Kurt@usa-spaceops.com

Roger Luty

United Space Alliance, LLC, USA
Roger.A.Luty@usa-spaceops.com

ABSTRACT

This paper describes the system engineering approach used to develop distributed multi-purpose simulations. The multi-purpose simulation architecture focuses on user needs, operations, flexibility, cost and maintenance. This approach was used to develop an International Space Station (ISS) simulator, which is called the International Space Station Integrated Simulation (ISIS)[1]. The ISIS runs unmodified ISS flight software, system models, and the astronaut command and control interface in an open system design that allows for rapid integration of multiple ISS models.

The initial intent of ISIS was to provide a distributed system that allows access to ISS flight software and models for the creation, test, and validation of crew and ground controller procedures. This capability reduces the cost and scheduling issues associated with utilizing standalone simulators in fixed locations, and facilitates discovering unknowns and errors earlier in the development lifecycle. Since its inception, the flexible architecture of the ISIS has allowed its purpose to evolve to include ground operator system and display training, flight software modification testing, and as a realistic test bed for Exploration automation technology research and development.

[1] The International Space Station Integrated Simulation (ISIS) is patent pending technology.

OBJECTIVE

This paper describes the system engineering approach used to develop a distributed multi-purpose simulation. Multiple instances of high fidelity simulations using real flight software and hardware to a distributed workforce are ideal. However, such simulations can be costly to duplicate or develop. Emulation of some system components to provide an acceptable distributed simulation can significantly reduce the resources required. Trade-off assessments are required to determine emulated versus unmodified components.

Our system engineering approach for ISIS focused on user needs and requirements, cost, operations, verification and flexibility to balance these trades. The primary goals were to develop a distributed multi-purpose simulator to extend the user community and to reduce scheduling issues associated with utilizing the Space Station Training Facility (SSTF). The International Space Station Integrated Simulation (ISIS) was originally developed to create, test, and validate crew and ground controller procedures. As a distributed simulator, ISIS facilitates discovering unknowns and errors earlier in the procedure development lifecycle.

BACKGROUND

International Space Station (ISS) Flight Controllers and Crew are responsible for operating and maintaining spacecraft systems. Understanding the interactions and complexities between systems, and the underlying flight software, is a core part of daily operations and training. Flight procedures are used to document commands and command sequences that must be executed in response to specific emergencies, as well as general maintenance and operations activities. The procedures contain commands and responses from the onboard flight software.

Flight controllers and mission planners develop, test and verify all procedures. The flight controllers and crew train extensively using these procedures. Training sessions are used to validate the procedures and any special conditions noted. After this validation, procedures are released for operations use. Access to the flight software is generally limited to a small number of facilities that are heavily used for integrated training and flight software verification.

GOALS

A need was identified for access to ISS system models and simulation in a distributed environment. Such a capability would reduce the cost and scheduling pressures associated with utilizing the primary ISS simulators in fixed and physically restricted locations.

Based on our discussions with users of the fixed base ISS simulator, we identified needs and requirements for ISIS. These

were used in developing the goals for the project. The system had to be economical to implement and maintain, and provide access to multiple users at multiple sites. The system had to allow anytime anywhere use of real ISS flight software and SSTF models and be expandable to accommodate additional models as ISS capability increases. The system had to be capable of running unmodified ISS flight software and integrating with existing ISS simulation models. The selected approach had to be easily adaptable to new or future spacecraft designs.

DEVELOPMENT APPROACH

After generating an overall concept, ISIS development was completed using the steps shown in Figure 1. This is an iterative approach to meet user needs and allow for growth. The steps can be categorized as: (1) develop project goals, identify capability of existing system, identify user needs, priorities and risks; (2) develop requirements; (3) Develop core architecture, design prototype and test with users, modify system if required; (4) Verify and implement system. Through understanding the capabilities of the existing system, user needs, priorities and inherent risks, the project goals were established.

Requirements were categorized by risk and priority, and selected subsets of the requirements were implemented. The highest project risks and the most critical system functionality and performance requirements were met first, before committing to development of the entire system. This approach allowed us to use a philosophy of "minimum commitment" in the initial development phases and

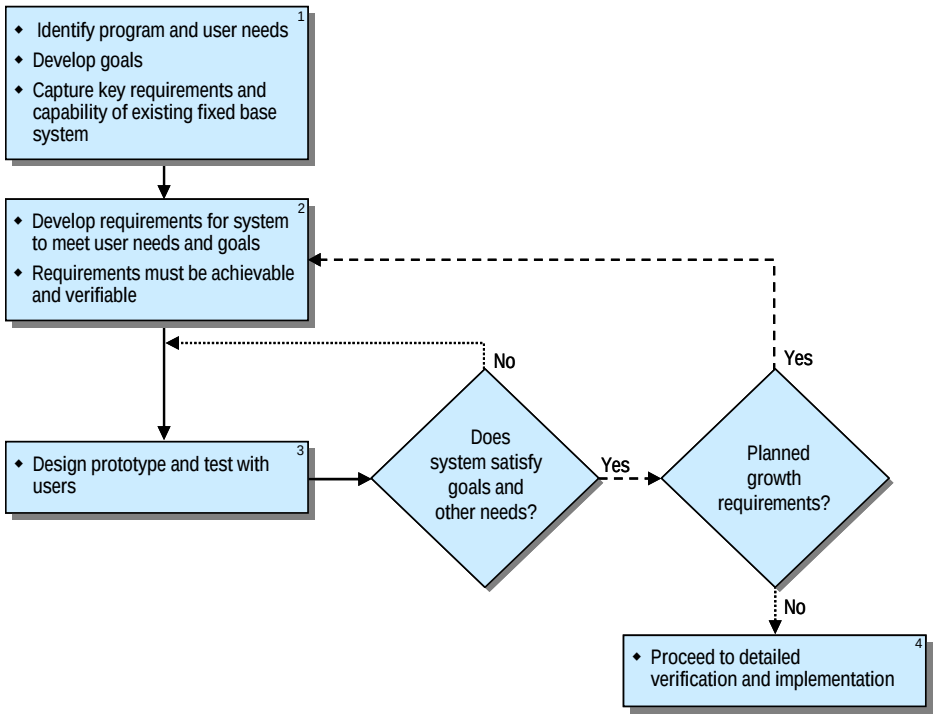

Figure 1. ISIS Development Approach. The project used a systematic, iterative and incremental approach to minimize resources required

allowed the project to analyze and build a robust and sustainable core architecture from several candidates, while managing cost.

We kept an architecture-centric focus and designed the initial system with modular components. The intent was to leave design options open as long as possible while technical risks were retired. As clear architectural choices emerged, the components were standardized.

A key focus in the system design was basing implementation design decisions on operational need. Emulation of the entire existing simulation environment was cost-prohibitive and beyond the requirements for our system. Complete emulation also introduced additional risk to the project, as our system would have lower fidelity in key areas.

Upon selection of the architecture, a prototype was designed and demonstrated to users to ensure goals were satisfied. An iterative process was used to mature the system and user feedback was continually received. Subsequently, the system was verified for implementation. Areas of planned growth were incorporated in a similar manner.

DESIGN DEVELOPMENT

Flight controllers maintain an extensive inventory of procedures necessary to safely operate the ISS. Knowledge of how the onboard Multiplexer/Demultiplexer (MDM) computers respond to each procedure pathway is required. The procedures are developed using existing ISS MDM flight software and component documentation, operational experience, peer reviews, and limited verification in training facilities prior to certification. The flight controllers identified the need for an ISS simulator to verify new and updated procedures using actual MDM flight software and high fidelity simulations of other components. This simulator would eliminate the need to utilize valuable training facility time to debug procedures. ISIS provides this capability.

We chose to implement the ISIS solution maximizing the use of available technology. Several products existed to carry out continuing development of the ISS, flight controller training and crew training. Evaluation of these products was limited to those using real MDM FSW. The users wanted to use the simulations developed for the Space Station Training Facility (SSTF) since these simulations are used for primary training and differences in fidelity between the actual ISS are well known. These simulations also come with an extensive set of malfunctions and an Instructor Operator Station (IOS) for their control.

The choices of available technology were narrowed down to two. These were the SSTF and an MDM Application Development Environment (MADE) developed to carry out Formal Qualification Tests (FQT) on MDM FSW. The key drivers influencing the selection were:

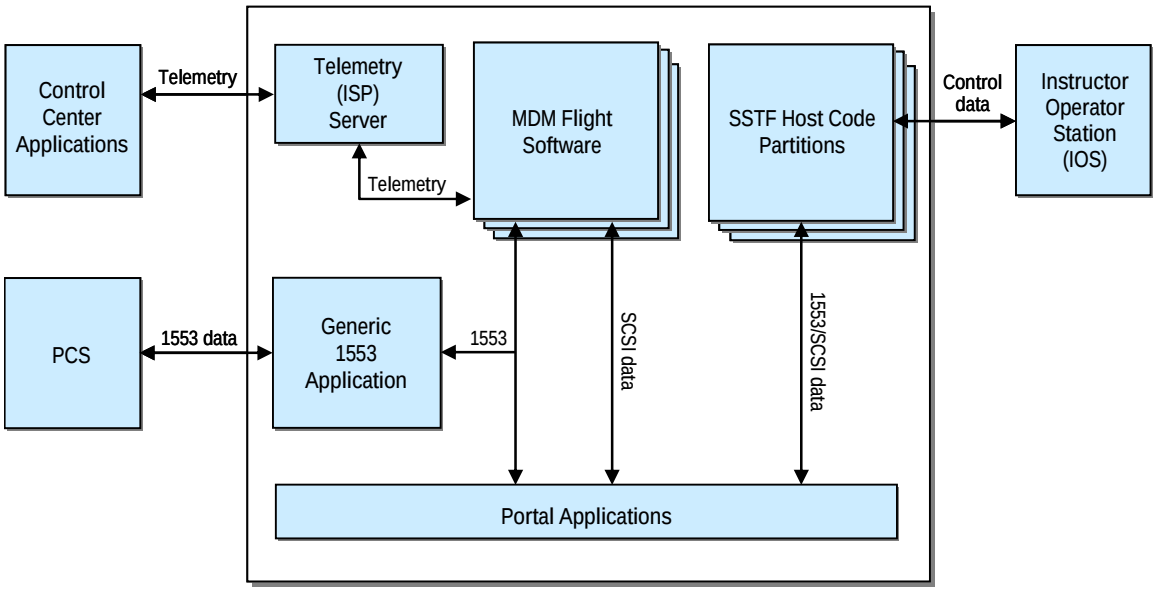

Figure 2. The open architecture of ISIS allows for rapid integration of multiple models throughout the life of the ISS

1. ISIS must run real unmodified MDM FSW to ensure high fidelity when compared to actual ISS operations.
2. ISIS should be controlled using an IOS-like application so users will not need extensive training on its use.
3. ISIS should utilize the SSTF simulation models to maintain the same level of fidelity and functionality the user community is used to.
4. ISIS must be able to drive existing MCC applications to provide users with familiar displays for ISS telemetry. These applications require their data be delivered to them via an ISP server and the data must be passed through front end processing to translate the raw telemetry into usable values.
5. ISIS must be able to support real PCS FSW to provide users with additional displays for ISS telemetry and a commanding pathway.

The solution that satisfied all of the key drivers was a hybrid SSTF-MADE system. The basic layout and dataflow of the ISIS is shown in Figure 2. The four key features shown are: the SSTF host code which runs the ISS software models, the MDM Application Development Environment (MADE) which runs the ISS MDM FSW, the telemetry server which drives the external applications (Information Sharing Protocol Server – ISP), and supporting applications that tie the system together. The external interfaces include the IOS, Portable Computer System (PCS), and Control Center applications. We gained an additional benefit from this configuration. Reusing existing components such as the IOS and Control Center applications in ISIS reduced or eliminated training time requirements for this "new" simulator.

CURRENT CAPABILITIES

The system is currently being used in a variety of ISS applications. They include flight controller procedure verification, flight software familiarization, and ESTL pre-test. The simulator is also being applied as a realistic test bed for NASA Exploration automation technology research and development including autonomous command string creation, verification and execution, and the integration of intelligent algorithms and tools for ground-based decision support for the Crew Exploration Vehicle.

Currently we are focused on using the simulator for part-task training of the crew and of the flight control team. New requirements are being analyzed to provide training-specific functionality.

SUMMARY

The systems engineering approach described here enabled the development of a new, multipurpose simulator, ISIS, that met user needs for a distributed and open design. Focusing on the overall operations goals ensured ISIS could coexist with current operations concepts. Using the highest priority requirements and risks to drive the design enabled the simulator to evolve without over designing the system or over committing resources.

The iterative, incremental development process allowed continuous interaction with the user community and quick turnaround on user feedback. Careful reuse of existing components helped to lower lifecycle costs, maintenance and training requirements for the new system. Finally, re-using communication standards and protocols that are common in the ISS environment instead of a custom solution allows ISIS to be extended to multiple ISS applications and domains.

ACKNOWLEDGEMENTS

The work reported herein was performed under NASA contracts NAS9-20000 and NNJ06VA01C.