Utilizing A Common Model Architecture For The Simulation Of On-Orbit Human Spaceflight Operations

Leslie J. Quiocho
NASA Johnson Space Center
Automation, Robotics & Simulation Division
Houston, Texas 77058, USA
leslie.j.quiocho@nasa.gov

Edwin Z. Crues, Ph.D.
NASA Johnson Space Center
Automation, Robotics & Simulation Division
Houston, Texas 77058, USA
edwin.z.crues@nasa.gov

John R. MacLean
METECS
Houston, Texas 77289, USA
jmaclean@metecs.com

ABSTRACT

Advances in computing and simulation technology during the 1980's promoted the development of multiple simulations for on-orbit human spaceflight operations at the National Aeronautics and Space Administration (NASA) Johnson Space Center (JSC). By the late 1980's, it became increasingly clear that redundancy risked leading to inefficiencies in dissemination and incorporation of models across simulations, institutional organizations, and spaceflight programs. Software development tools, such as the Trick Simulation Environment, began to emerge providing a generalized framework for the development, integration, and operation of simulations. Throughout the 1990's, engineering, operations, and training simulations began to migrate to Trick in order to take advantage of a common simulation environment.

Due to forecasted reductions in funding to the Shuttle and International Space Station (ISS) programs in the early 2000's, the common framework that arose from this simulation migration was formalized as the Common Model Architecture (CMA). The CMA was established in order to further reduce redundancy, while promoting model sharing and enhancing model integrity. A minimal set of standards for data flows, functional interactions, and model organization based on Trick were defined for those features found to be most important in facilitating model interaction. Model-providing organizations within NASA JSC took ownership of models for their specific areas of expertise, and agreements were reached on common model sharing and usage.

This paper describes the evolution of commonalty enabled by the Trick Simulation Environment at JSC leading to the CMA, the standards defined by the CMA, a recent CMA simulation example, and potential application of the CMA to NASA's new Exploration program.

Keywords:

Trick, spaceflight, commonality, architecture, exploration

1 BACKGROUND

The development of common model simulations at the NASA Johnson Space Center (JSC) has been a process of evolution. During the mid 1980's, both the engineering and mission operations communities used multiple simulations that had been developed independently. By the late 1980's, it became increasingly clear that this approach was redundant, and software tools, such as the Trick Simulation Environment began to emerge [1]. Trick provided a framework and a collection of utilities for simulation development, including a data-driven scheduling executive (non-real-time (NRT) or real-time (RT)), input processing, data recording, and Automatic Code Generation (ACG). Trick used file parsing and ACG techniques to generate input/output mechanisms as well as math model function calls, allowing the simulation developer to focus specifically on the task of math modeling and integration. Trick also supported simulation development across several versions of UNIX (e.g., IRIX, SunOS, and AIX); Trick is currently supported on several Linux operating systems and the Mac OS X operating system.

In the early 1990's, NRT engineering applications began to migrate to Trick in order to take advantage of a common simulation environment. Through this process, developers recognized the benefits of model reuse, as well as the shared leveraging of domain expertise provided through math models, particularly in the area of integrated simulation analysis.

Organizations began to assume responsibilities for providing models from their respective disciplines. For example, the Automation and Robotics Division provided remote manipulator system models for the Shuttle and Space Station manipulators as well as flexible multibody dynamics models [2], the Guidance, Navigation, and Controls (GN&C) Division provided Shuttle, ISS, and visiting vehicle flight control system models, while the Structures and Mechanisms Division provided docking and berthing mechanism models.

As one might expect, the availability of these models within the same, common simulation environment would provide inherent efficiencies not achievable while

working in a disparate sense. This is especially true when solving integrated spacecraft dynamics and controls problems involving multiple disciplines.

During the mid 1990's, RT engineering facilities began their migration to Trick. Examples include:

1. The Six Degree-of-freedom Dynamic Test System (SDTS), a Stewart platform used for testing of docking and berthing mechanism hardware (refer to Figure 1).

2. The Systems Engineering Simulator (SES), a crew-in-the-loop engineering simulator with high fidelity math models, scene generation, and cockpit mock-ups used for crew training and engineering evaluations.

3. The Interactive Control and Dynamics Simulation (ICDS), a SES-like desktop based simulator with a focus on Rendezvous/Proximity Operations (R/PO) between the Shuttle and the ISS.

With success achieved in conversion of engineering simulators to the Trick framework, operations and training facilities started their transition to Trick in the late 1990's. The following are some examples:

1. The Robotics Planning System (RPS) which includes kinematic simulations of the Shuttle Remote Manipulator System (SRMS) and Space Station Remote Manipulator System (SSRMS) used for mission design and procedures development.

2. The Multi-Use Robotic Manipulator Development Facility (MRMDF) which consists of a full-scale, 1g, hydraulic emulation of the SSRMS used for procedures and situational awareness training (refer to Figure 2).

3. The Neutral Buoyancy Lab (NBL) which consists of full-scale, underwater, hydraulic emulations of the SRMS and SSRMS manipulators used for robotics and Extravehicular Activity (EVA) procedures and situational awareness training.

4. The Dexterous Manipulator Trainer (DMT) which consists of a 1g, hydraulic emulation of the Special Purpose Dexterous Manipulator (SPDM) used for ISS maintenance procedures training.

5. The robotics segment of the Space Station Training Facility (SSTF), a fully integrated trainer used for procedures and integrated ISS mission training.

6. The Dynamics Skills Trainer (DST), a desktop crew familiarization and interactive skills proficiency trainer for SRMS, SSRMS, R/PO, and Shuttle Ascent/Entry.

7. The Virtual Reality Lab (VRL) trainer, a simulator used for integrated EVA/Robotics operations training in a virtual setting.

One relatively recent application of the Trick framework is avionics-in-the-loop simulators. Examples include 'hybrid' simulators for the Simplified Aid for EVA Rescue (SAFER), the Autonomous EVA Robotic Camera (AERCam), the X-38 GN&C system, and the Robotics On-board Trainer (ROBOT) which is used for crew refresher training aboard the ISS. Another recent area of application is distributed simulations with Trick using the High Level Architecture [3, 4].

What all of these simulations and simulators have in common is a framework for model development, integration, and operation that allows common model components and common technical experience to be disseminated and shared amongst facilities and organizations. This has led to dramatic and simultaneous improvements in efficiency while maintaining and even enhancing model integrity.

2 COMMON MODEL ARCHITECTURE

Driven by the need to capture lessons learned in the migration from disparate to common simulations, and in response to forecasted reductions in funding to both the ISS and Shuttle programs, an effort was initiated in 2003 to formalize the common framework that emerged from this evolutionary process. This activity resulted in a Common Model Architecture (CMA) that could be accepted across groups of model providers and users as a set of standards to guide development and utilization of common simulation models. Standards for data flow, functional interactions, and model organization were defined based on those Trick features found to be most important for facilitating model sharing and interaction.

Model providing organizations within JSC then took ownership of models for their specific areas of expertise and agreements were reached on common model sharing and usage. This reduced redundancy in model development and enhanced integrity of models as it co-located technical expertise and accountability for model validity.

2.1 Trick Simulation Development and Integration

Simply stated, the CMA philosophy has gained its widespread adoption through the use of Trick. Before discussing the architecture definition itself, it is therefore necessary to briefly describe some of the salient and relevant features of the Trick environment for simulation development and integration.

Trick has been described previously in [1]. It is comprised of a toolkit consisting of model development,

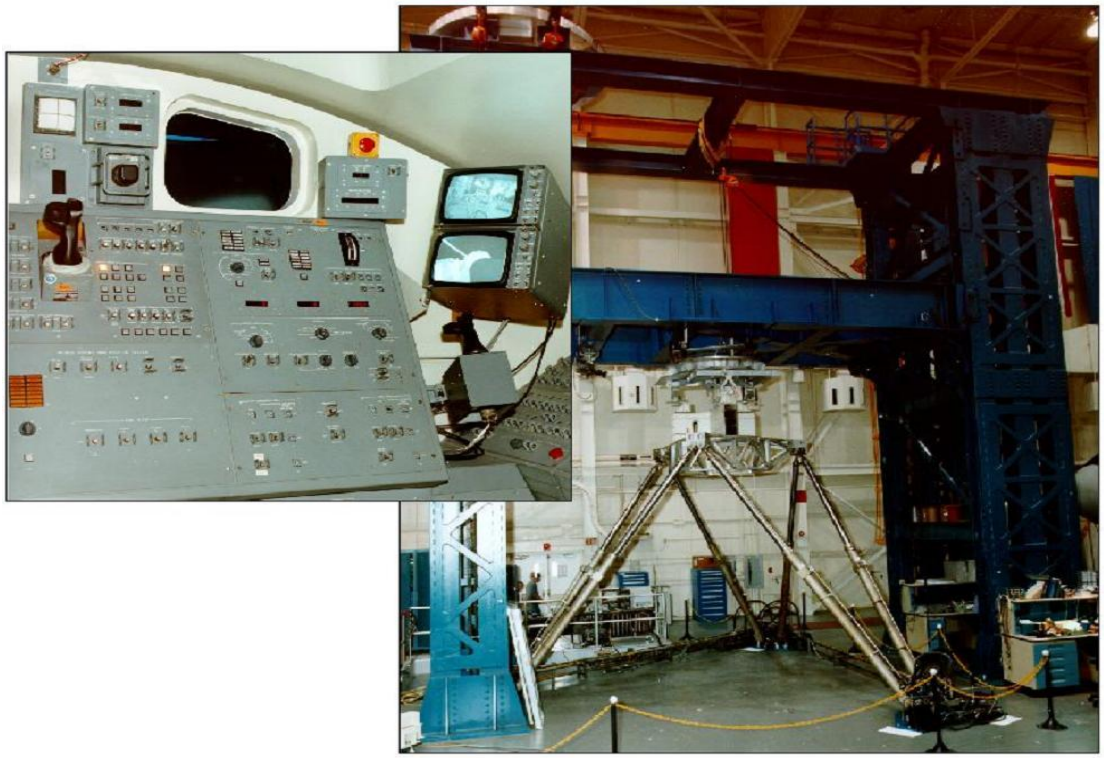


Figure 1: SDTS Docking and Berthing Simulator



Figure 2: MRMDF SSRMS Trainer

integration, scheduling, simulation control, and data analysis tools. The simulation itself is defined through a *S_define* file which is a C-like syntax text file with a format that defines simulation objects, the order of execution of the simulation objects, and the integration of the simulation object states.

Each *simulation object* in the S_define consists of data structure definitions and *module* or *job* definitions. The data structures are used to hold states and interface variables. The job definition includes a scheduling classification (e.g., initialization, derivative, integration, data logging, etc.), frequency of execution, interface parameters, and corresponding subroutine name.

To create a simulation executable, Trick parses the S_define file to determine what models are required to build that specific simulation application. It interprets the necessary model dependencies, creates a UNIX makefile specific to the defined simulation, and preprocesses and compiles any necessary model source code. It also creates a simulation executive that automates calling and integration sequences of objects and jobs as specified in the S_define file.

Each model interfaces with the Trick environment through the use of data structure definitions and job calling routines. These consist of C-like files with a well-defined syntactical structure to allow preprocessing by the Trick ACG utilities.

The Trick executive routines provide a set of simulation control flags for implementation of standard simulation capabilities such as freezing and data logging.

An in-depth description of the S_define file, the Trick executive, and the model interface syntactical requirements as well as other aspects of the simulation environment are provided in the Trick User's Guide [5].

2.2 CMA design standards

Most CMA design standards are based on what generally would be considered good programming practices. Some particular practices adopted pertain specifically to the C programming language, while others are strictly based on the Trick Simulation Environment. Below are the primary agreed to CMA design standards:

1. Data Mapping Standards
   (a) Data shall not be exchanged through global variables.
   (b) Data shall not be stored in local static variables, in order to prevent a model from being reentrant.
   (c) Data shall be contained and organized in structures, classes, or unions.
   (d) All data structures and classes shall have Trick compliant headers and field comments (as required by Trick for ACG).
2. Model Design Standards
   (a) All newly developed models shall be coded in the C or C++ language.
   (b) Models shall be reentrant.
   (c) All models shall be designed and implemented for full checkpoint restart capability (see Model Checkpoint Standards).
3. Model Calling Standards
   (a) Data (structures or variables) shall be passed into models through the model's argument list. For functions that use only a few parameters, variables should be passed as opposed to data structures, since this enhances reuse. Structures should be used to reduce the parameter list if many parameters are used inside a function.
   (b) Models shall not use global or external variables (see Model Design Standards).
   (c) Top level models (i.e., models called at the S_define level) shall be in the C or C++ language.
   (d) Top level models not in the C or C++ languages shall be called through C or C++ wrapper routines (see Model Wrapper Standards).
   (e) All model functions shall have Trick compliant header and function parameter list comments (as required by Trick for ACG).
4. Delivered Model Interface Standards
   (a) Externally developed models shall be delivered with Trick compliant interfaces (see Model Calling Standards).
   (b) Delivered models shall use the C or C++ language or the provider will supply C or C++ wrappers for calling the models (see Model Wrapper Standards).
5. Model Wrapper Standards
   (a) Delivered models not written in either C or C++ shall provide a set of Trick compliant interfaces written in C or C++.
   (b) Delivered models that do not have Trick compliant data types shall provide Trick compliant overlays for those data types.
   (c) Source code for all wrapped models shall be provided along with the associated Trick interface code.

6. S_define Organization Standards

   (a) All functions in the S_define file shall explicitly specify their job classification. The S_define file will not rely on a job's default classification.

   (b) All S_define data structures or instantiated classes shall have associated default data.

   (c) Vehicle avionics/flight software models shall be kept in separate simulation objects from the vehicle dynamics and environment simulation objects.

   (d) Vehicle interfaces to external I/O systems shall be kept in separate simulation objects.

   (e) Virtual scene generation capabilities shall be kept in separate simulation objects.

   (f) Communication routines with hardware components or external systems (e.g., custom user interfaces) shall be kept in simulation objects separate from the vehicle avionics/flight software models and dynamics.

7. Model Checkpoint Standards

   (a) Models shall provide all states necessary to support simulation snapshots when the simulation is in freeze mode as provided by the Trick checkpoint capability.

   (b) Models shall support the Trick simulation object specific checkpoint capability.

   (c) Models shall be capable of supporting a complete simulation restart from a full checkpoint.

8. Unit Conversion and Limit Standards

   (a) All simulation models shall have consistent units (i.e., either all International System of Units (SI) or all English units).

   (b) All data structures and function parameter lists shall include Trick compliant comments with appropriate unit specifications and data limits.

   (c) All default data files shall explicitly specify parameter units.

3 CMA SIMULATION EXAMPLE

A recent example of a simulation built using the CMA was one created to support Space Transportation System (STS) flight 114 (STS-114) Return to Flight (RTF) contingency Orbital Repair Maneuver (ORM) operations, as described in [6]. This NRT analytical simulation incorporated the reuse of several existing multi-discipline common models (developed by three different organizations). The simulated operation included undocking of the Orbiter from the ISS using the SRMS, maneuver of the Orbiter to a Thermal Protection System (TPS) repair position, EVA inspection and repair of the TPS using the SSRMS, and return and docking of the Orbiter with the SRMS. Common models reused within this simulation included: (1) control system and mechanism models of the Shuttle and Station manipulators, (2) flexible dynamics of the multibody chain topology, (3) orbital dynamics and earth environment models, and (4) an active ISS vehicle control system model (with attitude control and control moment gyros). The undocking and docking phase involving the SRMS and Orbiter docking mechanism system is discussed in [7]. Figure 3 illustrates the overall complexity of this integrated CMA simulation.

4 EXTENDING CMA TO EXPLORATION

As models have been developed, integrated and utilized in a coordinated fashion within the Trick environment to support both Shuttle and ISS Programs, it is fully expected that the architectural approaches can be applied to exploration and other future programs. As outlined in NASA's Exploration Systems Architecture Study [8], simulation opportunities will be numerous due to the large number of exploration assets. These assets include the Crew Launch Vehicle (CLV), Crew Exploration Vehicle (CEV), both Crew Module (CM) and Service Module (SM) components, Cargo Delivery Vehicle (CDV) for both unpressurized and pressurized cargo transport to ISS, and Lunar Surface Access Module (LSAM) to name a few. Each of these respective elements, which span the flight regimes of ascent, on-orbit, and landing, will require the substantial use of engineering simulations (both for NRT batch analyses and RT crew-in-the-loop), flight planning tools, and ground-based/on-board crew trainers.

5 CONCLUSION

The overall goal of the CMA approach through the use of Trick remains unchanged; identifying ways of simulating spacecraft systems efficiently while maintaining model integrity. The adoption of the CMA philosophy has been successful in supporting both Shuttle and ISS programs through engineering analysis, mission planning, and crew/flight controller training simulations. It is hoped that simulations and trainers will continue to capitalize on synergy and commonality in a fairly widespread manner, particularly with the emerging opportunities of the exploration program. CMA is poised to meet the needs of this new program by minimizing the development of duplicate capabilities and continuing to support NASA's role in human spaceflight.

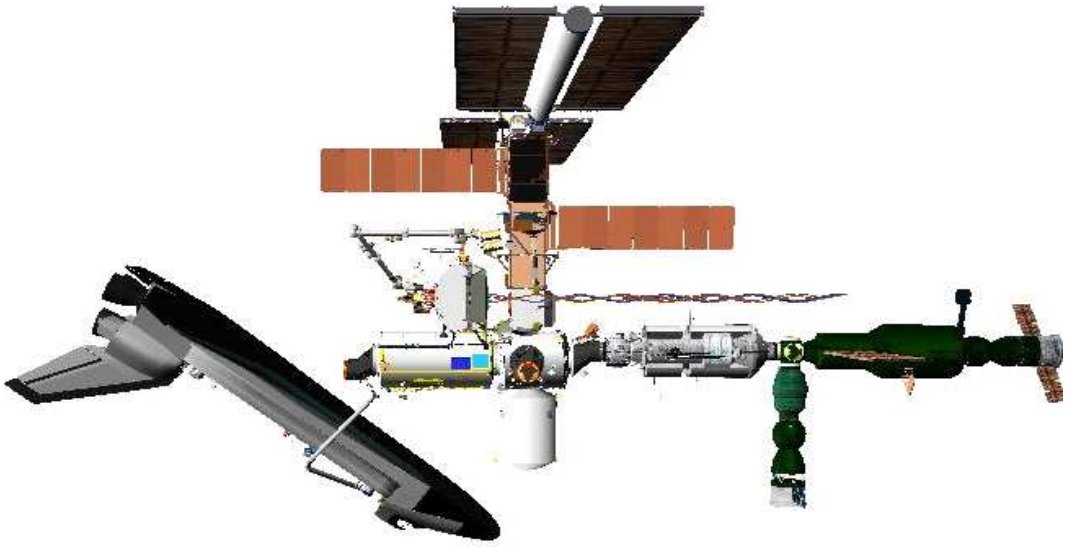


Figure 3: ORM Integrated Simulation Scenario

ACKNOWLEDGEMENTS

The work described in this paper was performed by the Simulation and Graphics Branch of the Automation, Robotics, and Simulation Division of the NASA JSC Engineering Directorate. The authors wish to recognize the following key individuals for their contributions to the CMA standards and advocacy of this approach in their respect disciplines: Bill Othon for vehicle GN&C systems and Tim Briscoe for docking and berthing mechanisms.

REFERENCES

[1] Paddock, E. J., Lin, A., Vetter, K., and Crues, E. Z., 2003. "Trick: A Simulation Development Toolkit." In Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit, AIAA 2003-5809, Austin, TX.

[2] Quiocho, L. J., Huynh, A., and Crues, E. Z., 2005. "Application of Multibody Dynamics to On-Orbit Manipulator Simulations." In Proceedings of the ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, DETC 2005-85545, Long Beach, CA.

[3] Crues, E. Z., Lin, A., and Hasan, D. A., 2003. "Integrating HLA into the Trick Simulation Development Toolkit." In Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit, AIAA 2003-5810, Austin, TX.

[4] Lauderdale, G. D., Crues, E. Z., Snyder, D. V., and Hasan, D. A., 2003. "A Feasibility Study For ISS and HTV Distributed Simulation." In Proceedings of the AIAA Modeling and Simulation Technologies Conference and Exhibit, AIAA 2003-5605, Austin, TX.

[5] Vetter, K., 2003. Trick User's Guide - 2003 Release, NASA JSC Technical Publication, Houston, TX.

[6] Quiocho, L. J., Crues, E. Z., Huynh, A., Nguyen, H. T., and MacLean, J. R., 2005. "Integrated Simulation Design Challenges to Support TPS Repair Operations." In Proceedings of the AIAA Guidance, Navigation, and Controls Conference and Exhibit, AIAA-2005-5988, San Francisco, CA.

[7] Quiocho, L. J., Briscoe, T. J., Schliesing, J. A., Braman, J. M., 2005. "SRMS Assisted Docking and Undocking for the Orbiter Repair Maneuver." In Proceedings of the AIAA Guidance, Navigation, and Controls Conference and Exhibit, AIAA-2005-5987, San Francisco, CA.

[8] NASA, 2005. NASA's Exploration Systems Architecture Study: Final Report, NASA-TM-2005-214062, Washington DC.