32st Space Symposium, Technical Track, Colorado Springs, Colorado, United States of America
Presented on April 11-12, 2016

SIMULATION-CENTRIC MODEL-BASED DEVELOPMENT FOR SPACECRAFT AND SMALL LAUNCH VEHICLES

Mike Briggs
MEI, mbriggs@meicompany.com

Nathaniel Benz
MEI, nbenz@meicompany.com

Douglas Forman
MEI, dforman@meicompany.com

ABSTRACT

The purpose of this paper is to characterize a form of Model-Based Systems Engineering (MBSE, (also known as Model-Based Development , or MBD)) as an integrated End-To-End (ETE) dynamic simulation-driven and code-generation-based process that facilitates and accelerates design and specification of complex systems. Inasmuch as a "scientific model" can be defined in a general way as a "physical or mathematical representation of a real phenomenon or object that captures its internal and external functions and behaviors to the extent and fidelity desired for the model use case", a suitable general definition of MBD is "composition and use of a scientific model to specify designs for and assess behavior and performance of a complex system of subsystems". These definitions provide a framework for accomplishing MBSE as we know it today.

In this paper, we primarily address applying MBSE to support development of complex systems that are composed of (1) sensors to measure system states, (2) actuators to apply forces or torques to the system or its components in response to commands, (3) data processors with embedded software to estimate states from sensor outputs and generate actuator commands, (4) system external inputs and operating environments, and (5) system outputs/products or payloads. Although these five entities are sufficiently general to represent a wide variety of complex systems, we choose to emphasize Spacecraft and Space Launch Vehicles as good examples of such systems within this paper to illustrate a modest functionality and complexity subset of the much larger spectrum that can be addressed using MBSE. The examples shown herein exploit visual charts and executable block-diagrams provided by commercial design-automation products to represent physical attributes and dynamics of physical systems comprised of hardware and software functions, the flow of logic/control and data and automated generation of source code from specifications provided by the block diagrams and information flow.

This paper will address MBSE in the following approach. The Introduction discusses the motivation for applying Model Based Systems Engineering and provides some insight into the variety of methods and tools that have been promoted for application to MBSE. Section 2 discusses processes applicable to MBSE and history of its development. Section 3 describes design-automation tools chains for small and large projects, and Section 4 discusses techniques for block-diagram programming of hardware and software models. Automated Code Generation (Section 5) addresses automated code generation and compilation targeted to operating systems or middleware, and is followed in Section 6 by a discussion of verifying and validating simulation models for prediction of system performance and generating standards-compliant robust and error-free code for embedded systems. The paper concludes with descriptions in Deployment and Operation of how the same system simulation created across the development phase is modified to support operational deployment.

INTRODUCTION

The motivation for adopting ETE MBSE is to provide substantial improvements in the quality of complex products while reducing the time and expense required to develop, deploy and support them. The paper provides insights into those metrics in the context of low-cost navigation, guidance and control avionics for spacecraft and small launch vehicles. Clearly, the effectiveness of this process is dominated by the fidelity of the underlying dynamic system models, so the paper provides examples of detailed subsystem modeling and simulation including effects of uncertainties and sensor measurement errors, and addresses real-time simulation issues associated with system verification and validation.

Although a variety of methodologies have been characterized as desirable MBD processes, including "Model-Based Systems Engineering", "Model-Based Development", "Model-Based Software Development", and other additional descriptors that invoke the term "Simulation-Driven" in place of "Model-Based" terminology, many of these address only part of the problem, such as focusing strictly upon software development.[1] [2] [3] or only the left (definition) side of the V-chart (e.g. Figure 1). Other methods that claim MBD capability describe behavior using software tools that automate Specification and Description Language (SDL) diagrams such UML and SysML[4] [5]. Although SDL diagrams can be effective in documenting and communicating proposed functionality descriptors, architectures and dataflow across diverse teams, the fact that they are currently incapable of simulating behavior makes them an incomplete solution that must be augmented through integration with simulation-centric, design-automation tools such as Simulink. This provides means of performing dynamic simulations of integrated system hardware and software to support iterative design-and-test as well as simulation-based verification and validation, and illustrates some of the need for an integrated chain of design automation tools as will be discussed in Section 2 (Model-Based Systems Engineering) of this paper.

MODEL-BASED SYSTEMS ENGINEERING

Systems Engineering is an interdisciplinary approach and a means to enable successful development of "systems", which are recognized as being composed of interdependent, interacting elements that act in concert to satisfy user needs and requirements. Systems Engineering is then recognized as an end-to-end scheme for organizing and controlling the application of engineering techniques to the engineering of complete systems in a manner that recognizes the interdependence and interactive nature of a system's elements throughout the life cycles of systems. Systems Engineering also integrates all the necessary disciplines and specialties needed to develop a specific system into a team effort, forming a structured development process that, in principle, proceeds seamlessly from concept to production to operation and disposal, actual experience notwithstanding.

Figure 1 shows the ubiquitous "V-diagram" provided in the form shown in Chapter 4 of the Defense Acquisition Guidebook[6] to illustrate the systems engineering process at the top level, representing top-down requirements definition and analysis, design and specification followed by bottom-up implementation, verification, validation, deployment and operational support. From this diagram, it is clear that Systems Engineering encompasses a lot more than requirements analysis, decomposition and flowdown and is, in fact, an End-To-End (ETE) process that also guides the physical realization and deployment of systems. The definition of Model-Based Systems Engineering presented earlier (composition and use of a scientific model to specify designs for and assess behavior and performance of a complex system of subsystems) clearly encompasses the ETE requirement for systems engineering.

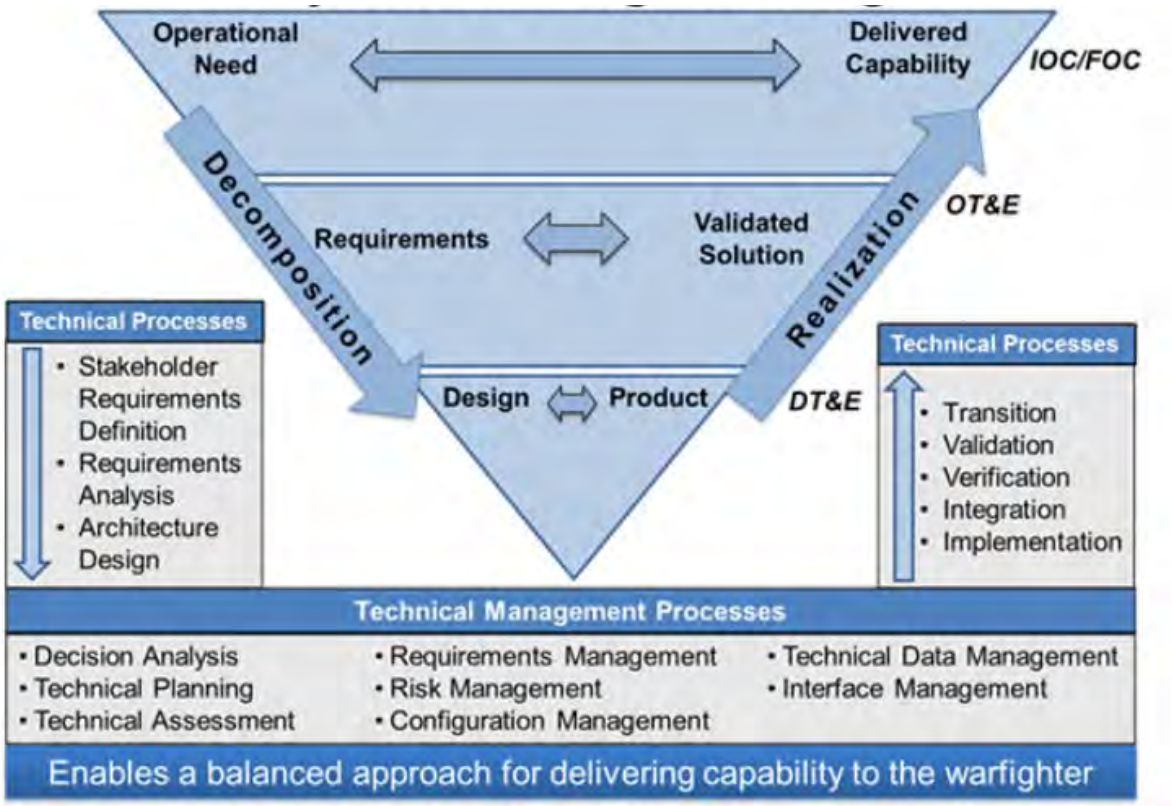

Figure 1: The End-To-End Systems Engineering Process as Represented in Chapter 4 of the Defense Acquisition Guide

As a result, the term Model-Based Development (MBD) is therefore synonymous with Model-Based Systems Engineering (MBSE), and can be defined as: "Composition and use of a scientific model to specify designs for and assess behavior and performance of a complex system of subsystems". As a result, the abbreviations "MBD" and "MBSE" are used interchangeably in this paper. The ETE MBD approach emphasizes simulation-driven functional and performance testing and verification at every step in the "V-diagram", as per the following activities that parallel those shown in Figure 1:

1. **Requirements Analysis & Flowdown**:, Emphasize cognitive identification of functions as candidates for satisfying requirements, modeling and dynamic simulation of requirements-compliant functions and function parameters, and allocation of the functions and requirements to subsystem models so as to constitute, test and identify viable architectures;
2. **Analysis of Alternatives:** Model dynamic simulation and performance assessment of subsystem alternatives to support and verify trade studies;
3. **Subsystem Requirements Specification:** Extraction of subsystem models and parameters from the simulation to prepare subsystem requirements specifications that support their procurement or manufacturing, with maintenance of the simulation as a life-cycle source of subsystem functional and performance specifications;
4. **Subsystem Model Verification:** Reconciliation of simulation model outputs with subsystem test data to facilitate constructive simulation validation;
5. **FMEA/FMECA, Risk Assessment and Mission Assurance:** Simulate failure modes and effects to define risks, evaluate potential risk mitigations and provide simulated data to support Mission Assurance activities;

6. **Embedded Software Generation:** Automate generation of source code for embedded software from the simulation targeting compiler/operating system/processor combinations or middleware, with seamless compilation linking and download to test targets;
7. **Test Driver & System Test Case Generation:** Auto-generate real-time plant-model code to provide simulation-drivers for Processor-In-Loop and Hardware-In-Loop tests of systems. Perform dynamic simulation of ground and flight tests to support test planning and provide a basis for comparison of achieved vs. expected test results with reconciliation of the simulation models to test data;
8. **Verification & Validation (V&V) Using Formal Methods:** Automate support of verification and validation by processing the simulation models and generated code through software-based theorem provers, model checkers, static code analysis & standards compliance analysis tools;
9. **Operational Training and Command Verification:** Integrate the system simulation with operations-compliant user interfaces to provide means of training operators and also provide means of testing and verifying safety and effectiveness of operator commands before they are applied to the actual system; and
10. **Operational Anomaly Resolution:** Use the system simulation to perform system anomaly assessment, including fault identification, isolation and duplication via simulation, and dynamic simulation-based testing and verification of fault accommodation actions. This activity also applies to simulation-directed preventative maintenance.

Figure 2 illustrates integration of several of these ETE dynamic simulation, model-based activities into the V-chart workflow, wherein the iterative nature of interactions between modeling/simulation and feedback of changes derived from assessment results is depicted.

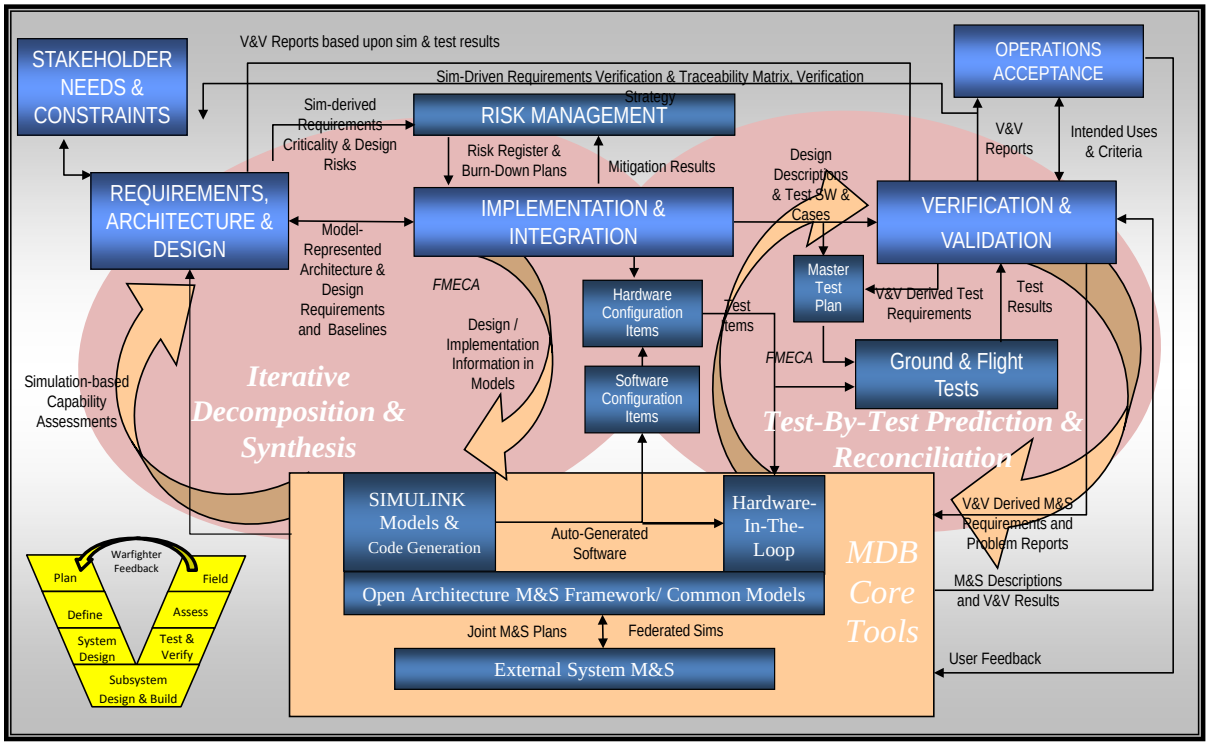

Figure 2: Representing Model-Based Development Within the ETE Systems Engineering V-chart

The "Iterative Decomposition and Synthesis" feedback loop depicted on the left-hand side of Figure 2 is shown in more detail in Figure 3, which also shows how dynamic simulation is applied to accomplish the four basic tasks shown.

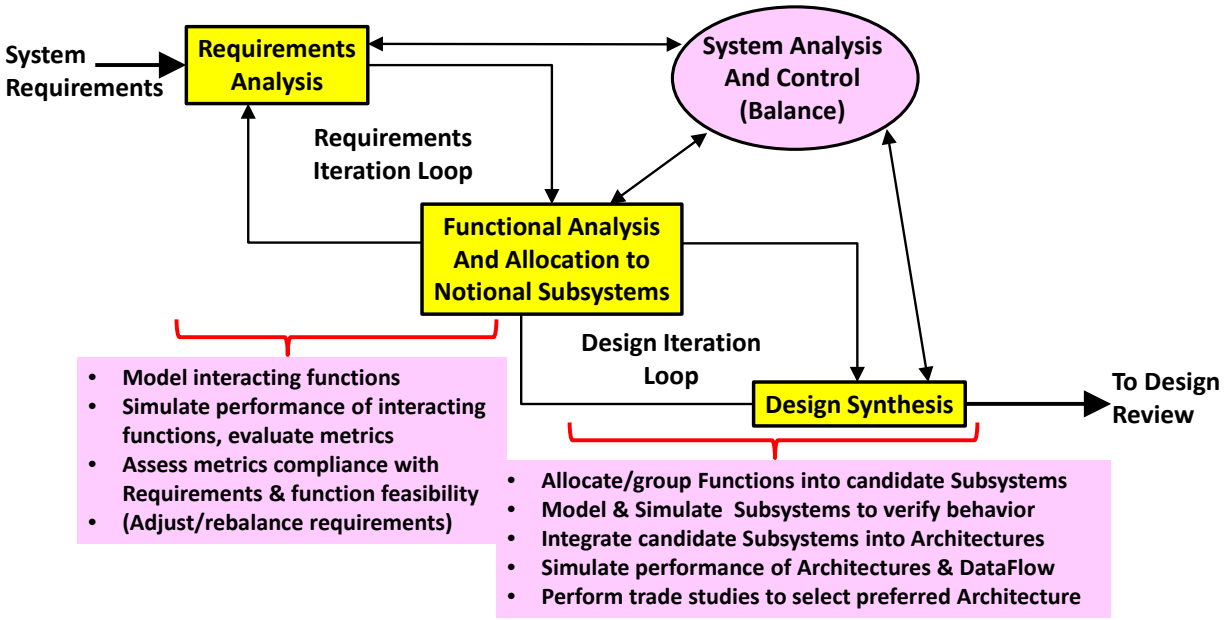

Figure 3: Simulation-Driven Iterative Decomposition and Synthesis[7]

History of Model Based Development

Model-Based Systems Engineering (MBSE) has evolved as term used in the Systems Engineering lexicon over the past 20 years or so to be descriptive of a transition from "Document Centric Systems Engineering" (the old way) to "Model Centric Systems Engineering"[8]. The ideas behind system science and engineering and the role of "Models" has been around since 1925, but the effort required to develop computer-based system models, the advent of government procurement of complex systems and the need for design traceability and understanding of it by diverse users shifted the focus to "document-centric systems engineering" from the 1960's to the mid-1990's.

This document-centric process began to evolve in the 1960s, but prior to that systems theory was built upon modeling and simulation. For example, a "General Systems Theory" was presented by biologist Ludwig von Bertalanffy in lectures as early as 1937, publications beginning in 1946 and ultimately a 1968 book[9], wherein von Bertalanffy maintains that a system is characterized by the interactions of its components and the nonlinearity of those interactions rather than the sum of actions of a system's components, and "integrating Philosophy and Theory as Knowledge, and Method and Application as action, Systems Inquiry then is knowledgeable action" that can be represented by a theory or Model. In a 1956 issue of "Management Science"[10], K.E. Boulding begins by stating "General Systems Theory is a name which has come into use to describe a level of theoretical model-building which lies somewhere between the highly generalized constructions of pure mathematics and the specific theories of the specialized disciplines", clearly establishing models as the driving 1950's thought regarding General Systems Theory clearly emphasized models as the driving force behind systems engineering in the 1950's.

Decades later, when block-diagram programming and Universal Modeling Language (UML) automation tools matured in performance, reliability and acceptance in the late 1990's/early 2000's, the availability of these tools on desktops and laptops (rather than limited-access mainframes) and their demonstrated ability to increase software reliability, communication of design intent and reduce development and operation cost led to a resurgence of models and dynamic simulations as the foundation of Systems Engineering, now known as Model Based Systems Engineering (MBSE).

UML and SysML have been emphasized[4 5] as tools-of-choice to implement MBSE, but they must be integrated with design-automation tools such as Simulink that provide means of performing dynamic simulations of integrated system hardware and software in order to provide a basis for evaluating functional allocations and designs. This is a result of the fact that most SDL tools were developed to focus only on the definition-side of Systems Engineering and therefore avoid addressing the remainder of systems engineering represented by the right-hand side of the ETE systems engineering V-chart (see Figure 1).

UML evolved from approaches to software design developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in 1994–96. Construction and specification of UML and SysML diagrams and charts are automated in IBM's Rhapsody product as well as COTS products from other companies wherein users can create all of the diagrams and charts specified by the UML and SysML standards using drag/drop and point/click actions. The UML and SysML features of Rhapsody and the several other COTS software packages that implement these standards do not provide indigenous means of simulating behavior of dynamic systems; however, the frameworks provided allow users to write and insert code segments into the user-defined architecture elements represented in their UML/SysML diagrams, and most of the COTS tools provide means of generating code for arbitrary architectures that embed the user-defined code segments. If UML/SysML diagrams are used to specify the behavior of system hardware elements whose state derivatives depend upon time and parameters, then such time-dependence must be represented via discrete integration. Since this can be time-consuming and subject to user errors, UML/SysML models are seldom used to prescribe dynamic simulations; in fact, most users of UML/SysML tools rely on external dynamic simulation frameworks to design coupled hardware/software systems, and then either write or generate code modules for the software side of the design to insert back into architectures represented in UML/SysML diagrams and charts. The user can then use the code-generation features of Rhapsody or other COTS tools to generate code for tasks or complete architectures.

Applying UML and SysML in a Model Based Development Process

Millennium prefers to use UML/SysML tools such as Rhapsody to define software architectures using Structure Models, Object Models, State Charts, Activity Charts and Sequence Diagrams for coordination across teams, and simultaneously create "Plant" (system hardware) models in Simulink. Figure 4 depicts the process used to generate Simulink subsystem architectures from Rhapsody diagrams and charts using a conversion tool we call RhapLink. We apply XMI-based conversion (using the MDA-sponsored RhapLink tool) to generate Simulink block diagrams that correspond to the UML/SysML source software diagrams, models and charts. Once the software architecture and I/O is fully represented in Simulink, we then design the system software within each auto-generated Simulink

subsystem to operate the Plant model, using dynamic simulation to verify compliance with requirements. Code generation from Simulink then provides the basis of hardware/software integration, test and verification.

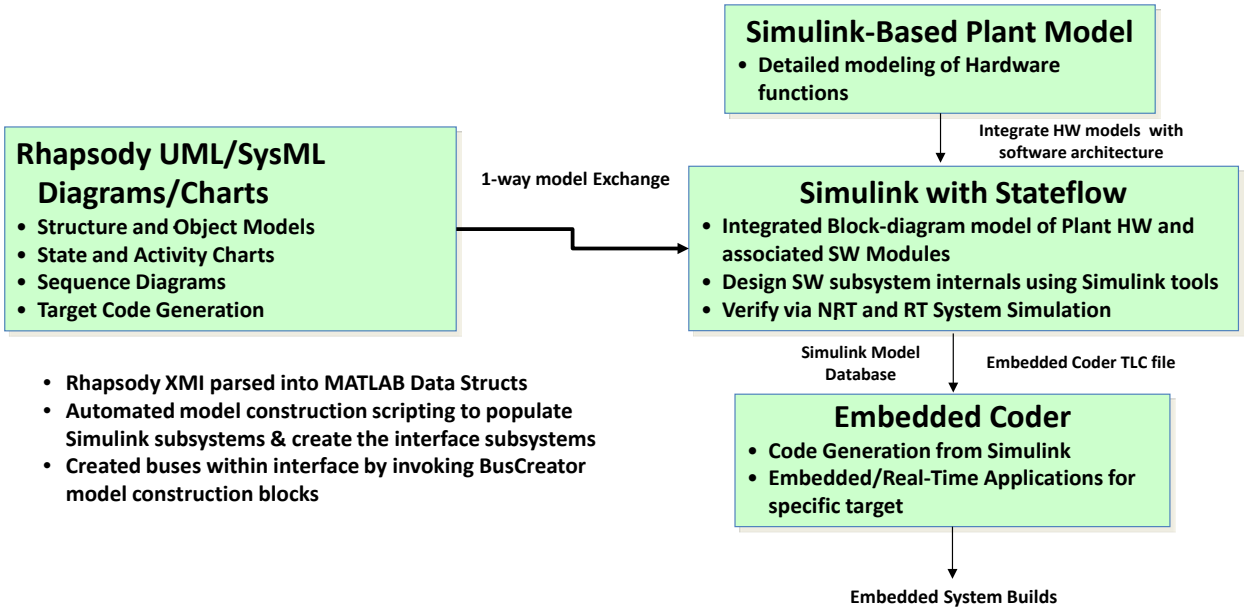

Figure 4: Converting Rhapsody UML/SysML Diagrams & Charts into Simulink Models Using RhapLink

The evolution of capability in the tool-chains suitable for effective MBSE to their current state occurred over more than 5 decades, wherein development of computer-aided engineering tools for research in control systems and simulation was under way in the 1970's (e.g., MATLAB by Cleve Moler). This led to development and commercial release of the first visual block-diagram programming tool, MATRIXx/System Build (by Integrated Systems, Inc.), in the mid-1980's, followed by Autocode in 1988, Simulink (by The Mathworks) in the mid- 1990's and Simulink's Embedded Coder in 2003. The late 1990's saw evolution of the maturity and capability of block-diagram programming coupled with automated code generation into powerful and reliable systems engineering and development tools that were adopted for complex programs such as ISS (a million lines of Autocode-generated code onboard embedded in life support, GN&C and experimental subsystems), DC-X, F-18 E/F, F-35 and others.

In summary, we strongly advocate that dynamic simulation is a prerequisite capability for ETE MBSE because simulation is essential for testing of concepts and designs to evaluate performance in the absence of a physical realization of the system. If the system has dynamic elements in which behavior is time-dependent, then dynamic simulation is required. However, we also strongly believe that use-case-adaptive integration of requirements tracking, configuration management, SDL diagramming, and block-diagram high-level programming tools with code-generation, middleware, V&V and hardware-specific targeting tools is essential to support the breadth and depth of ETE MBD of complex systems.

TOOL CHAINS FOR MODEL BASED DEVELOPMENT PROCESS ALTERNATIVES

Accomplishing Model-Based Development using the top-level process defined in Section 2 requires a minimum set of computer-aided engineering software tools to perform the following minimum set of basic process functions:

- Collect and organize system requirements in a manner that facilitates identification of and traceability to functions, subsystems and components. Requires use of a spreadsheet or a commercial requirements management product such as DOORS.

- Perform architecture design and modeling of hardware subsystems and software functions of the system using block-diagram programming software tools such as MATLAB/Simulink with Stateflow, or although other tools such as SCADE, VISSIM, SAFEPROG, SCICOS, Damos or LabView. This process function is supported by re-use of legacy block-diagram models (re-parameterized for use in the new block diagram model the user is developing), as well re-use of legacy C or C++ source code modules.
- Define and implement a process for configuration management, version control and build management. This is facilitated by selection and application of an automated software tool such as SubVersioN (SVN), Jira, Visual source safe, Concurrent version system, Rational Clear Case, or one of the dozens of such tools that are commercially available as open-source. Millennium uses SVN for most internal work, but adopts whatever the customer may prefer.
- Select an automated source-code generation software tool that is compatible with the block-diagram modeling tools selected above and generates source code in the required or desired language. Simulink with Embedded Coder is Millennium's preference, although several other tools are available including ones that generate code from Simulink. Most code generators (coders) for embedded real-time software are designed to generate C code, but code generation templates are generally available to apply the changes required to convert it into C++. ADA language code generators are available for Simulink and SCADE.
- Select or define a target processor board based upon allocated performance, memory and I/O requirements, and also select or create an Integrated Development Environment that supports the target processor with compiler, linker/loader, onboard debugger, profiler, board-support package and real-time operating system.

Figure 5 depicts what we consider to be a minimum of eight entities that must be integrated as seamlessly as possible (given budget and tool availability and feature sets) to support ETE MBD by providing an entry-level automated "menu-selection-driven" software development and targeting system. This class of software development systems is suitable for small projects wherein only 2 or 3 engineers are involved in systems engineering, design and implementation. In this case, most of the engineering effort is expended in use of the block diagram programming (MATLAB/Simulink) tool-chain component to define functional behavior flowed down from requirements, allocate functional models to subsystems to define architectures including I/O, and simulate system performance to assess compliance with requirements. This follows our basic rule of modeling for MBSE: "If a system contains dynamic (time-dependent) elements, then dynamic simulation is required to represent its behavior". Another significant part of the small-project effort must be devoted to tailoring the code-generation process to target elements of the Real Time Operating System (RTOS) portion of the Integrated Development Environment so as to provide a rapid, seamless path from block-diagram models to embedded code running on the selected target processor. This has to be accomplished one time at the outset of code-generation activities, and then religiously maintained and updated as needed over the life-cycle of the system.

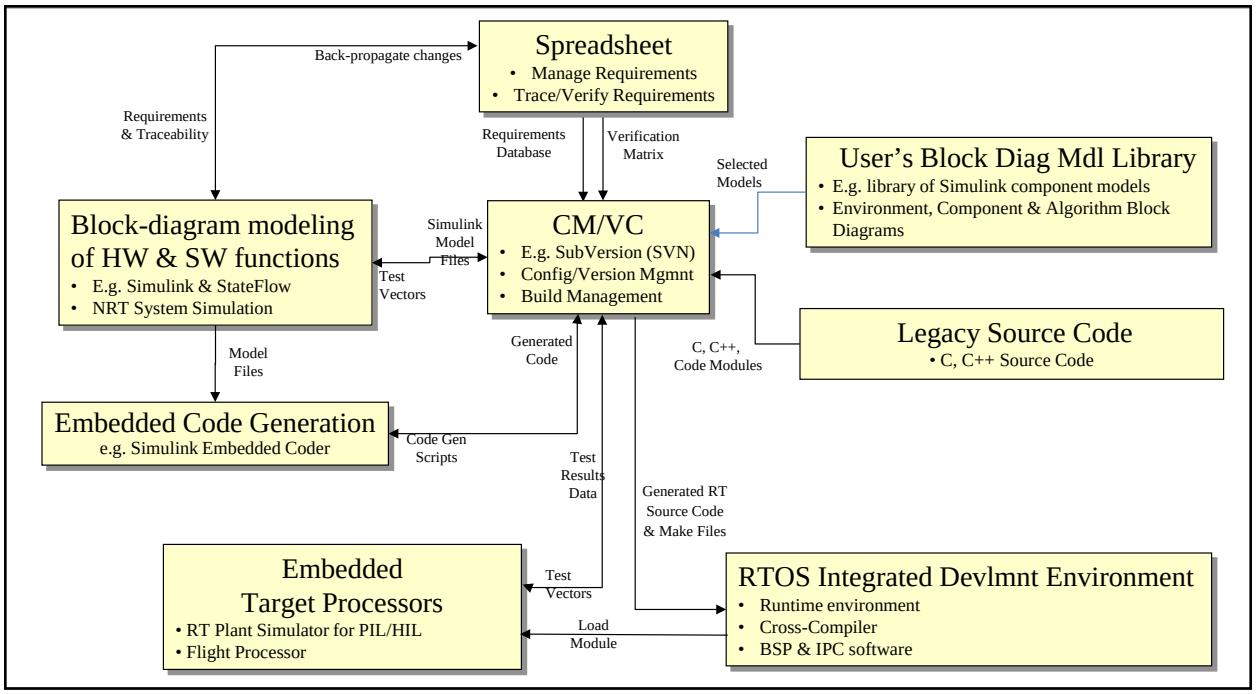

Figure 5: Model Based Software Development Environment for Small Teams

For larger, more complex systems that require a team of (about) 4-5 or significantly more engineers to accomplish systems engineering, design and implementation, it is often desirable to incorporate Middleware (such as Core Flight System (CFS) with Core Flight Executive (cFE) now available as open source by NASA Godddard Spaceflight Center) to provide a more loosely-coupled software system that greatly reduces development, integration and testing complexity for large systems with contributions from multiple developers, agencies and/or companies.

Also, the requirements definition and flow-down, architecture definition, analysis and concept definition phases of programs that are applying a team of engineers to develop a complex system can become chaotic unless task-specific standards are rigorously applied to coordinate and communicate hardware and software architecture across the team. Such a standard is available from the Object Management Group (OMG) which specifies a UML for visual specification, visualization, and documentation for models of software systems. In addition, the International Council on Systems Engineering (INCOSE), in collaboration with the OMG, has created general purpose visual modeling language for systems engineering applications (SysML). Private companies have developed software-based automation tools that facilitate creation of the several diagram and chart types specified by the UML and SysML standards, which range from passive architecture and activity diagrams to executable statecharts.

To seamlessly integrate middleware as well as UML/SysML functional and data specifications into an MBD environment, data translation software entities are required so that:

1. Architectures and statecharts defined in a commercial UML/SysML product such as Rhapsody can be seamlessly and accurately transferred into a simulation-capable block-diagram-programming product such as Simulink, as discussed in Section 2 (see Figure 4). To provide this capability, the Missile Defense Agency sponsored Millennium's development of RhapLink -- a Rhapsody-to-Simulink translator -- that uses Rhapsody diagrams and charts represented in its internally-generated UML-compliant metadata exchange (XMI) files to generate Simulink models composed of Simulink Subsystem Blocks, buses and single-element data connections.

2. Code can be generated as middleware "tasks" that seamlessly integrate into middleware products such as Core Flight System. The software "glue code" to accomplish this was developed cooperatively by NASA Ames Research Center (with Millennium's support) and The Mathworks, Inc., makers of Matlab.

The tool chain depicted in 6 reveals an MBD environment for complex systems being developed by moderate-to-large teams of engineers. It illustrates seamless integration of Requirements Management and UML/SysML diagrams and charts as well as generation of CFS tasks from Simulink, and also shows integration of Physical models to capture component-specific behavior (e.g. multibody dynamics and/or servoelastic behavior.

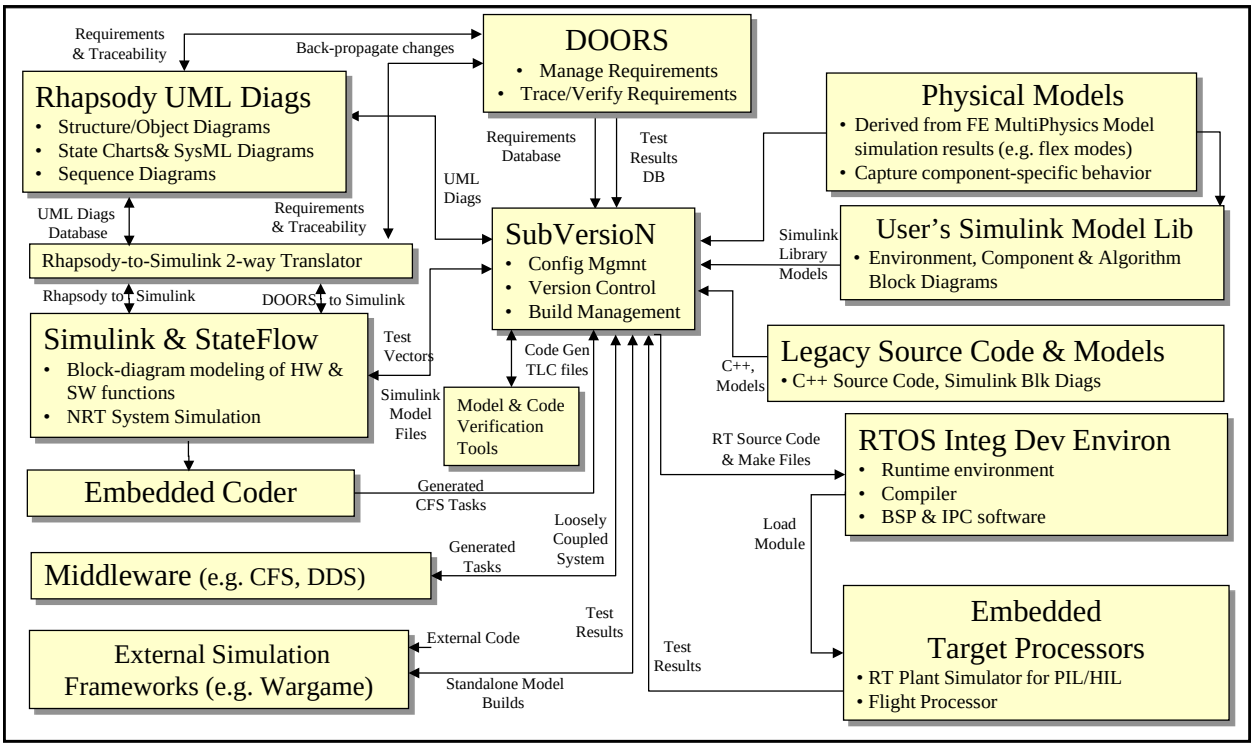

Figure 6: Model Based Software Development Environment for Large Teams Developing Complex Systems

A component of the tool-chain has been added to represent "External Simulation Frameworks", wherein simulations may be automatically exported via code generation to larger-scale simulation frameworks such as discrete-event frameworks used to simulate wargames. The Target Language Compiler (TLC) capability of Simulink Embedded Coder enables automated parsing of generated C-code to insert directives and adaptively instantiate new context-dependent operations, thereby enabling users to develop custom model-exchange tools. This level of integration was achieved by exploiting the open data-interchange features of the specific products such as is depicted in Figure 4.

BLOCK-DIAGRAM PROGRAMMING AND SIMULATION OF HARDWARE AND SOFTWARE MODELS

Graphical models are used throughout the engineering disciplines: mechanical engineers create computer-aided design (CAD) models, electrical engineers produce circuit diagrams, software engineers create flow charts, and controls engineers draw block diagrams to represent their respective systems. This section is focused on use of block-diagram programming to create models of subsystem hardware and embedded software models. Simulation is applied at the functional unit and subsystem levels to verify that design intent and requirements are satisfied, and the subsystem models are ultimately tested and reconciled against actual subsystem test data. The subsystem

models are frequently tested in integrated form to accomplish testing and evaluation of system performance to assess compliance with system requirements.

One advantage of MBD is that control system engineers are able to rapidly prototype algorithms in their natural design environment with tools such as Simulink and software engineers can directly auto-code the models. This minimizes the likelihood for communication errors between algorithm designers and software developers. The model-based methodology also enhances early prototyping of requirements, enables validation and verification during early stages of development and provides a common platform for communication between subsystems, software engineers and stakeholders.

Figure 7 shows a generalized control system diagram which consists of five main components: a controller, actuators, system, sensors, and an estimator. As shown in Figure 7, a desired state and estimated state are combined to form the state error which is input into the controller. The controller manipulates the signal and sends a command message to a system (also known as plant) model. The system model represents the system being controlled such as a spacecraft or rocket and contains dynamic models of the system's actuators and sensors. The measured state of the plant is fed to the estimator which combines all measurements into a new estimated state.

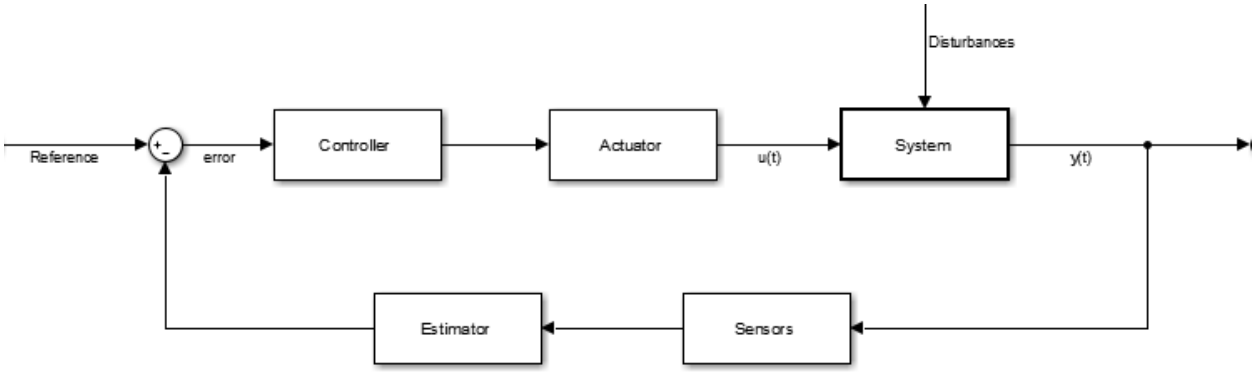

Figure 7: Generic Control System Block Diagram

Example Modeling

Models of the plant are created by creating a mathematical model of the system dynamics and then implementing that model into a block diagram. As an alternative to deriving the mathematical equations, engineers can also create a model represented in the physical domain using Simulink add-on toolbox Simscape. The process is best demonstrated through an example.

Mechanical system are often model as analogous mass-spring-damper systems. For example fuel slosh in a rocket or the suspension of planetary rover.

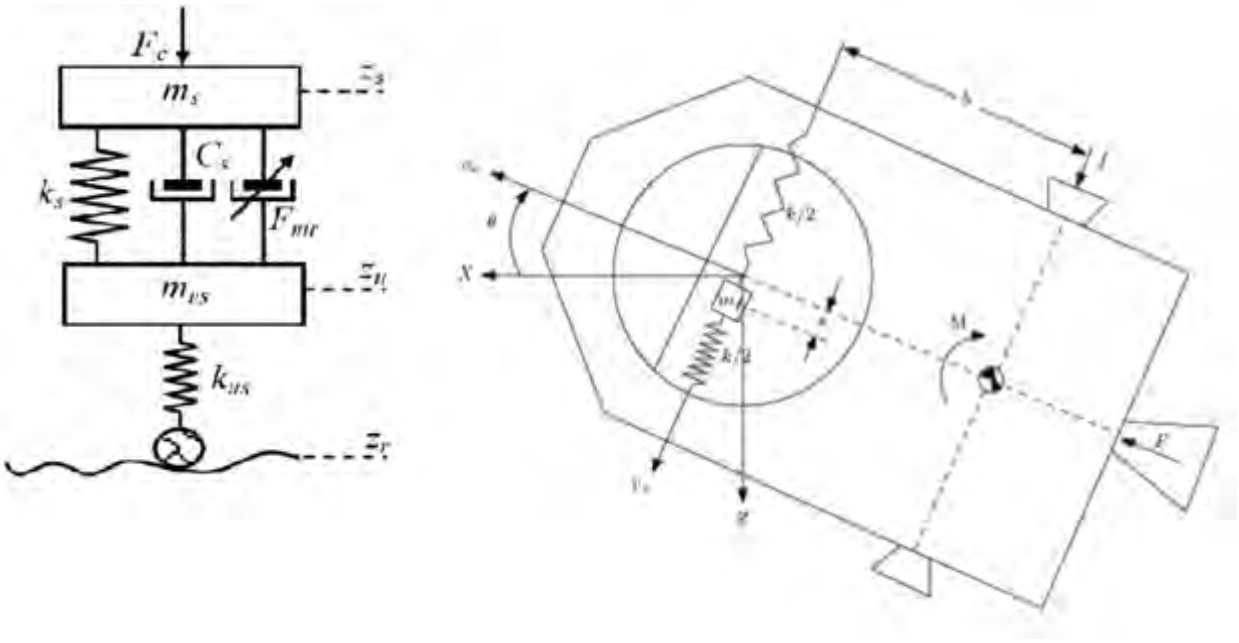

Figure 8: Example of Mass-Spring-Damper System Modeling

The mathematical equations for such mass-spring-damper system can be derived and then implemented in a block diagram or physical representation can be represented in Simscape.

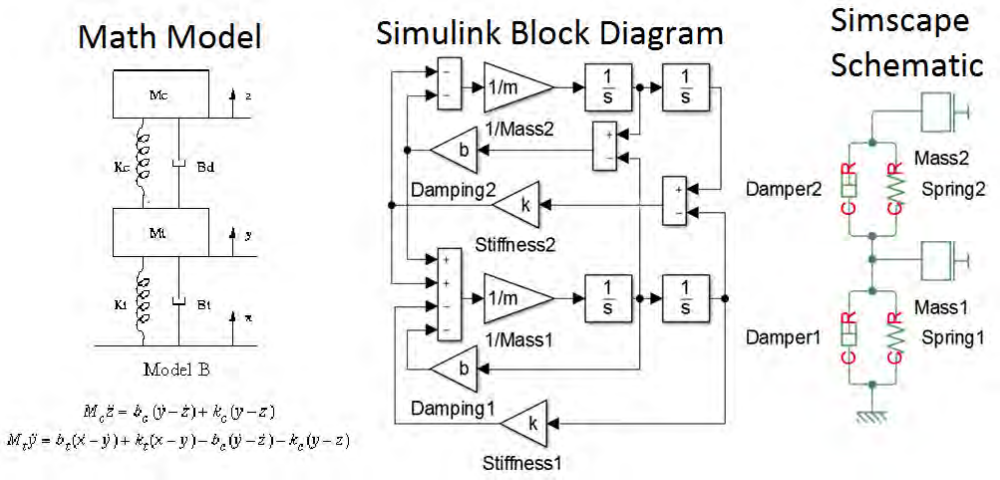

Figure 9: Modeling Using Simulink and Simscape

The two systems from Figure 9 represented in Simulink and Simscape are identical and can be simulated within the Simulink environment. Simscape provides the advantage of a schematic that is much easier to identify system relationships. Pure Simulink blocks have an advantage in simulation speed over Simscape blocks.

Importing Models from CAD Software

An additional method of creating plant models in the physical domain is the use of SimMechanics (an add-on to Simscape). SimMechanics allows the user to import an assembly model from CAD software such as SolidWorks directly into Simulink. The model import maintains the physical attributes of individual components (e.g., mass,

inertia) as well as the constraints between components (e.g., revolute, translational joints). As an example, Figure 10 shows a robotic manipulator CAD model developed in SolidWorks that was imported into Simulink via the SimMechanics link. A controls engineer then instantly has a plant model for use in simulation and design for the manipulator controller.

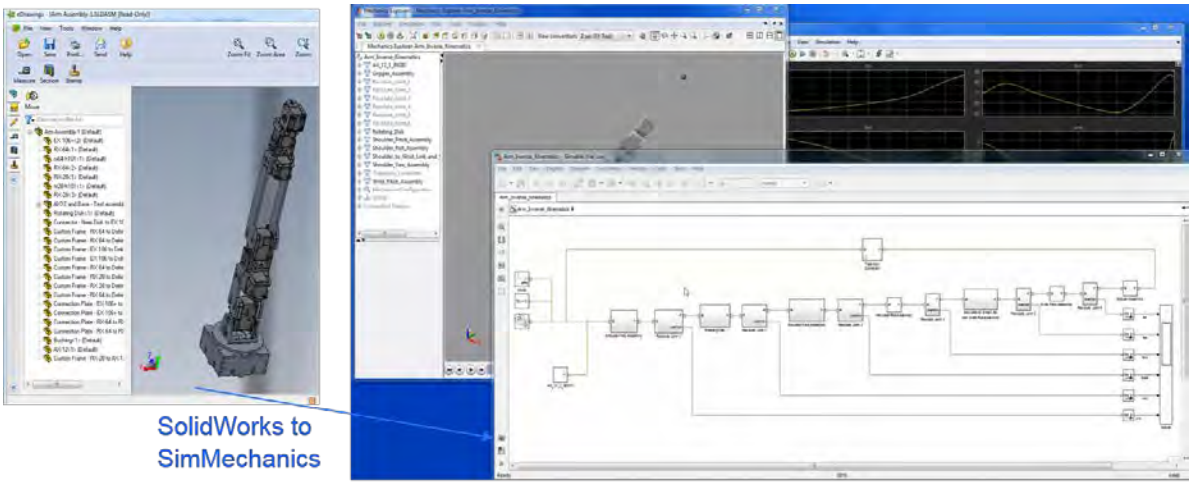

Figure 10: Importing CAD models into SimMechanics.

Case Studies

LADEE Spacecraft

As was established in previous sections, modeling and simulation constitutes the principal activity of Model Based Systems Engineering, and the application examples shown in this section are based upon the development of the Lunar Atmosphere and Dust Environment Experiment (LADEE) spacecraft and its simulator. Development of the LADEE flight software utilized a MBD approach that integrated auto-generated code, minimal hand-code (e.g. for communications drivers), Government-off-the-shelf (GOTS) middleware and Commercial-off-the-shelf (COTS) computer-aided engineering software packages[11]. Most of the on-board flight software as well the simulation representing the spacecraft physical subsystems and software is modeled using MATLAB/Simulink from The Mathworks, Inc, wherein test systems are generated from the plant model and concurrently flight software is generated from the command and control simulation models, wherein the system simulation is the single source of generated code.

The model based-methodology enables prototyping and testing of functional/algorithm sequences that directly address requirements in support of functional analysis and allocation to subsystems, providing an unprecedented level of confidence in the evolving design iterations. It enables validation and verification during early stages of development and provides a common platform for communication between subsystems, software engineers and stakeholders[12]. A key advantage of this process is the common usage of the graphically-programmed models to accomplish multiple purposes. For example, LADEE control system engineers were able to rapidly prototype algorithms and then simulate and verify them in the MATLAB/Simulink design environment followed by handoff of the verified models to software engineers who would then directly generate code from the models. This minimizes communication errors between algorithm designers and software developers.

The process depicted in Figure 12 was applied to develop subsystem models that were then integrated to form the LADEE System Simulation, which is comprised of the Spacecraft simulation with its embedded software, and the Ground Control Station “simulation” with its embedded software. These two simulation entities were initially constructed in MATLAB/Simulink and code generation was used to create Processor-In-Loop and Hardware-In-Loop test-simulation entities.

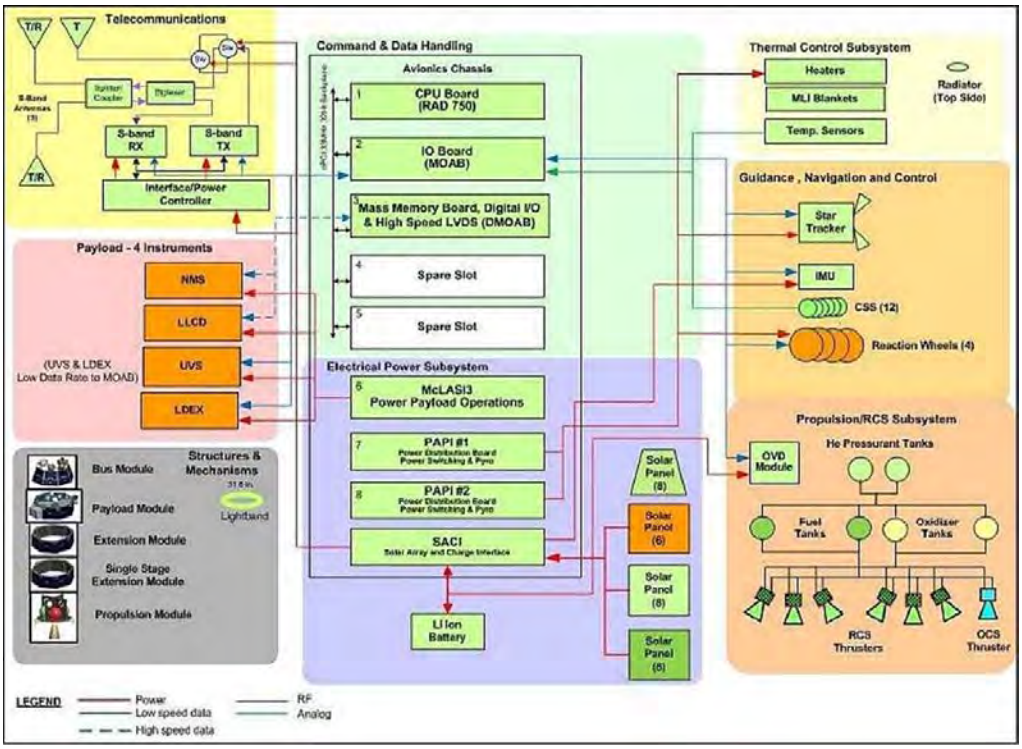

Figure 11: LADEE spacecraft functions and subsystem components.

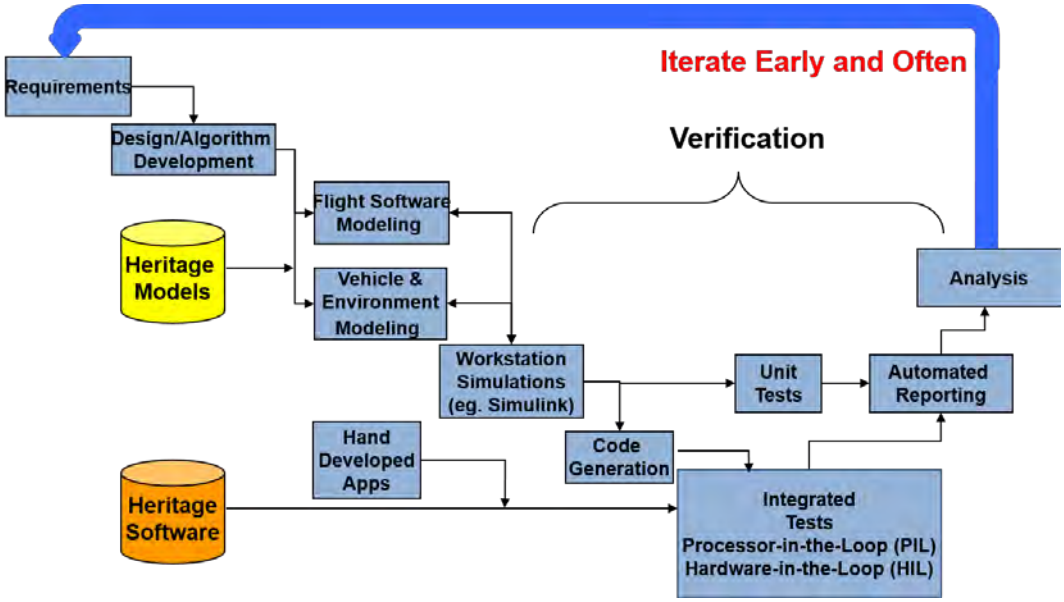

Figure 12: Model-Base Development Process on LADEE

Low Cost Space Launch Vehicle

In order to provide affordable acces to space for small spacecraft, NASA and the DoD investigated the use of a common and modular avionics platform to provide Guidance Navigations and Control (GNC) to several proposed low cost space launch vehicles. Model Based Design was used to carry out trade studies on the use of lower cost sensors GNC. High fidelity models of the prosed Inertial Measurement Unit (IMU) and GPS sensors along with environment models (NASA's GRAM wind model) and an Extended Kalman Filter were simulated with Monte Carlo runs to characterize the dispersion envelope of the proposed design.

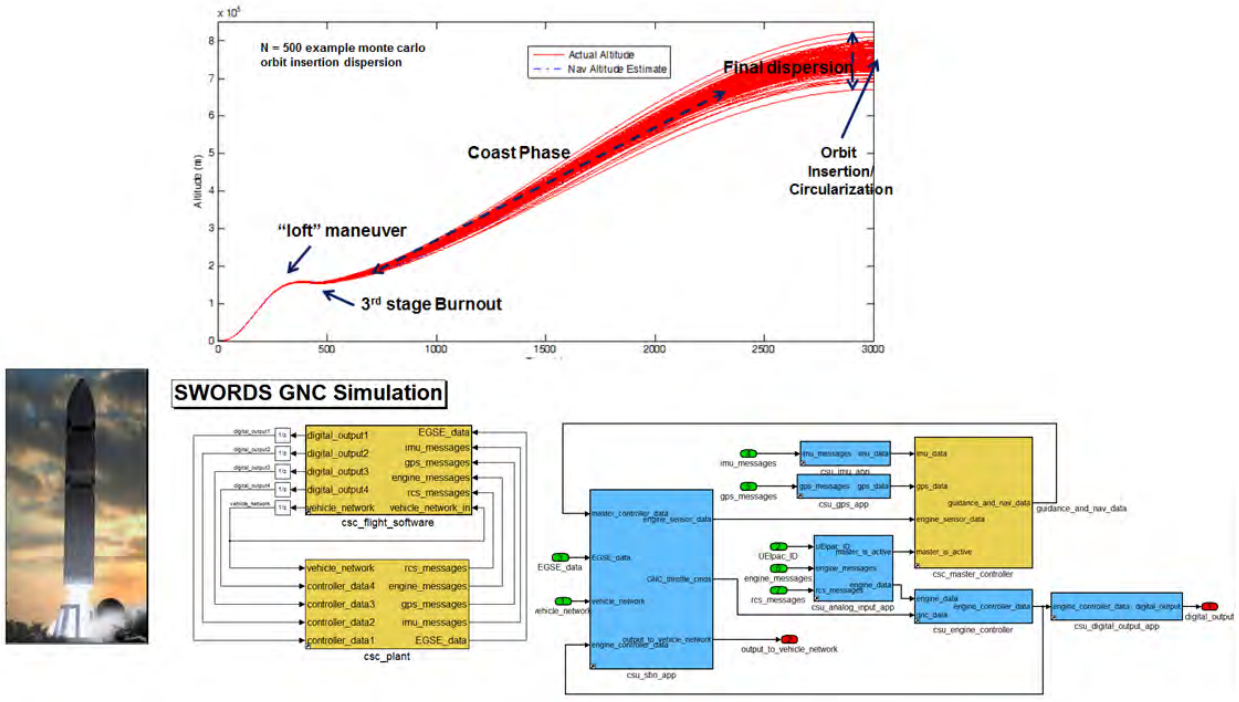

Figure 13: Low cost launch vehicle simulation and trade studies.

Telerobotic Remote Servicing

Model Based Design was also successfully demonstrated in the area of spacecraft robotics. In order to remotely control and service existing satellites a concept of a tele-operated robotic manipulator with haptic feedback and latency compensation was design. The model based solution utilized a feature in Simulink called S-Functions to import legacy C/C++ code to model the inverse kinematics and scene generation of the manipulator. The concept was integrated and proven on the Naval Research Laboratory Robot Arm.

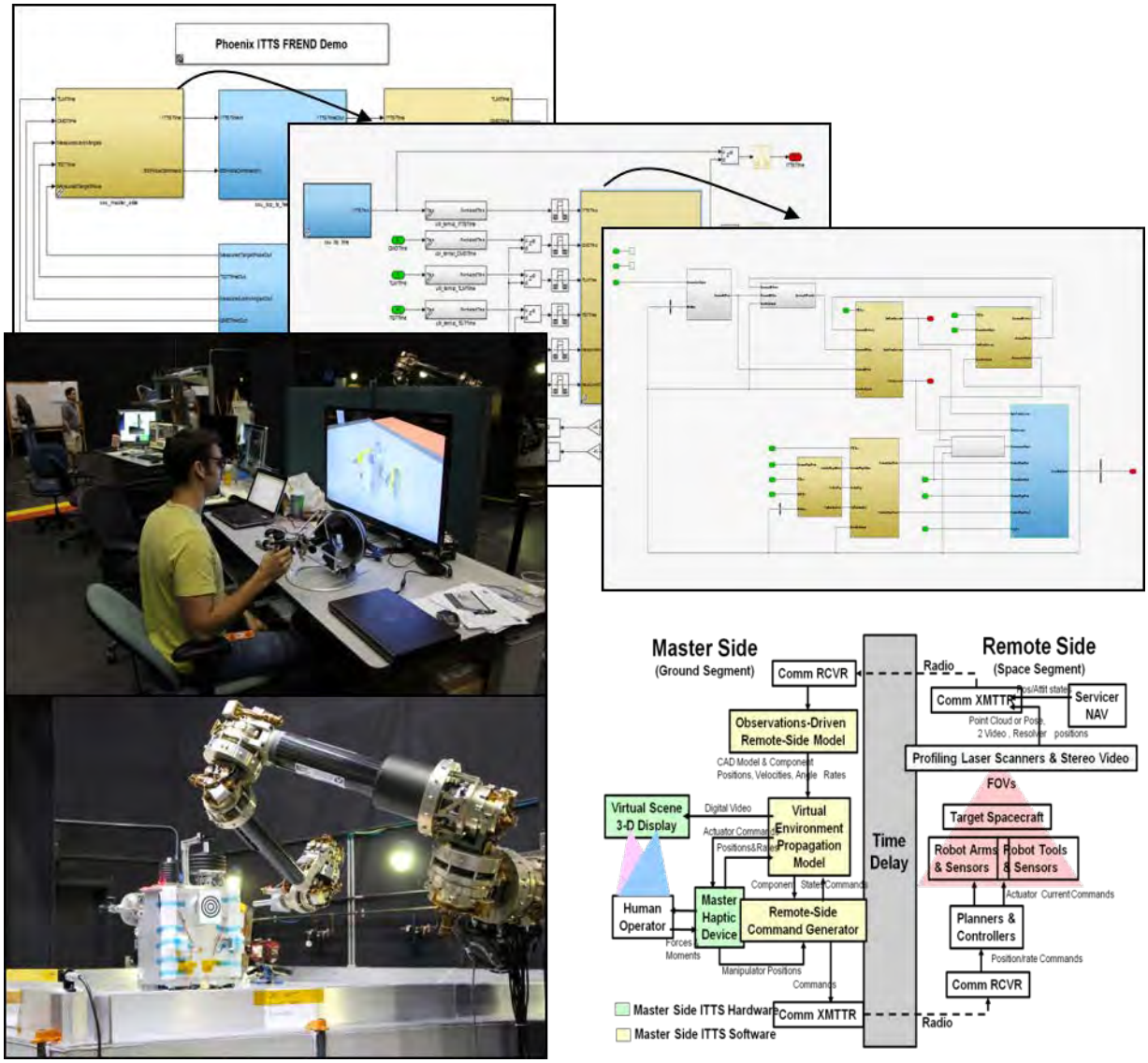

Figure 14: Telerobotic Manipulator modeled and demonstrated.

SIMULATION AND SOFTWARE MODEL TESTING

In general the later in the development process a software bug is found the more expensive it is to fix. For this reason, an early emphasis should be placed on the simulation unit test suite infrastructure with an eye toward permanently capturing the unit test effort as a regression test suite. In parallel to the unit testing, individual components should be combined into an integrated system model to simulate and test the high level system requirements. Requirements defined in external source (DOORS, SysML, spread sheets) can be directly link in the model test suite to create requirements traceability. The traceability links can be used to validate all requirements have a test case and identify missing or untested requirements. The requirement link in the model also gets inserted as a comment in the generated code. In order to prove complete coverage of defined requirements Formal Methods tools can be used to identify modeling errors and automatically generate test vectors to reproduce requirement violations in simulation. Lastly static code analyzers can be used on the generated C code to prove the absence of

run-time errors before being deployed to hardware. The MBD testing process can be summarized in Figure 15

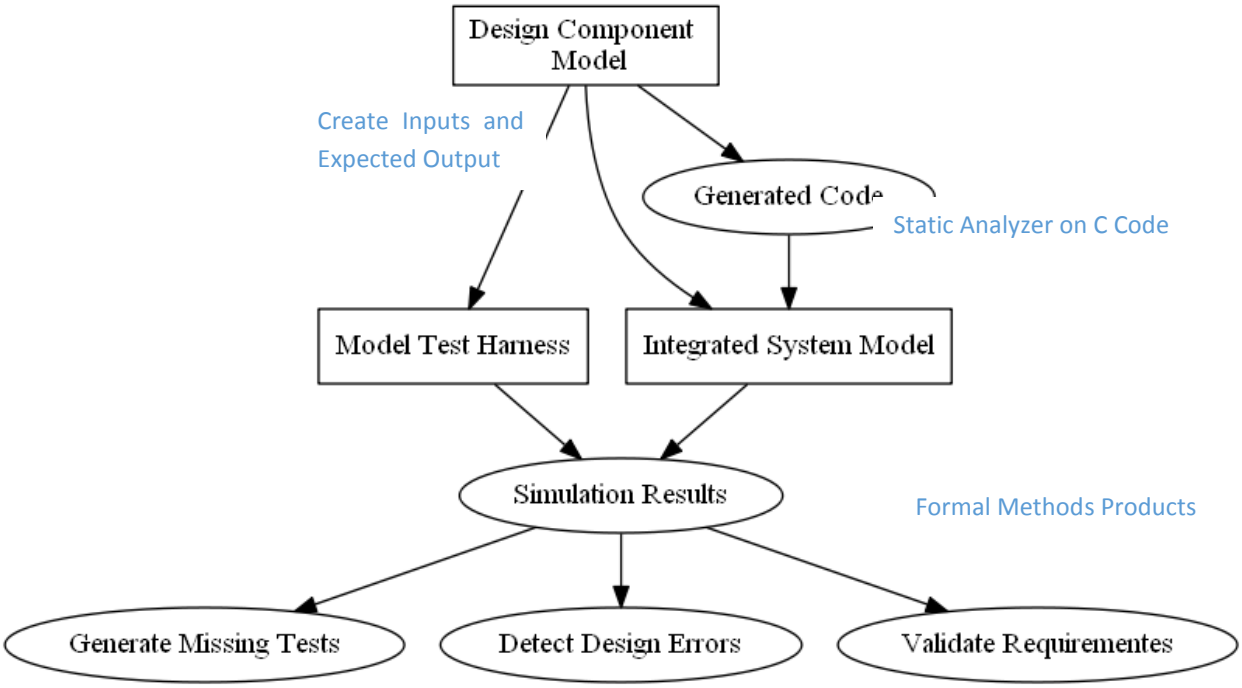

Figure 15: Model Based testing steps.

Spacecraft Example

The development effort for the NASA LADEE spacecraft separated testing into two different types: unit tests and system integration tests. Low level requirements were tested at the unit level because it is usually hard to test and debug internal modes and interfaces at a system level. Higher level requirements were tested with an integrated model because it is impossible to test system integration issues at a unit level.

Testing on the LADEE program was enhanced by the modular and layered system architecture. To maintain bidirectional traceability between code/models, requirements, design and test artifacts a system of naming conventions was enforced. For example, for the modeling environment, pertinent naming conventions are:

- {name}_lib.mdl: Simulink Model library
- {name}_hrn.mdl: Simulink test harness
- {name}_test.m: Test script associated with model library.

Where the {name} included a unique identifier for that model library for cross- referencing with requirements and other external documents. Each developer was responsible for providing all the necessary artifacts and developing the unit test suite associated with their model libraries. By adhering to the naming conventions, higher-level regression test suites could interrogate the LADEE model and generate a report of test artifacts. This report was used to exercise the entire test suite under development and evaluate the progress on verifying requirements. Simulink Report Generator was used as a platform to both drive the test suites and capture the resulting information in a manner that provided bi-directional traceability between low-level requirements, models, test suites and metrics.

This test suite also exercises the unit tests on simulator side of the LADEE model. Assumptions are documented as well as ranges of acceptable operation for the model. Depending on the type of model, refinement studies, sensitivity analyses, comparison with analysis, or subsystem performance specifications are included.

The unit test suites are used as early indicators that pertinent requirements are being met, but they do not test the integrated functionality of the system and its software. That is done in a Processor- or Hardware-In-The-Loop

(PIL/HIL) test environment using scenario-based testing. A spacecraft concept of operations was developed that looked at each phase of the life cycle and scenarios were developed to test anticipated operations and occurrence of faults. These scenarios included separation and activation, science operations, orbital maneuvering and fault management related scenarios.

Once the scenarios have been executed, a custom Simulink report generator script is used to post process the data and capture the output. The script first reads external imported data, such as requirements spreadsheets and Interface Control Documents. It then processes each of the data files to extract needed variables for the test suite[10].

AUTOMATED CODE GENERATION AND COMPILATION

This section of the paper addresses automated code generation from Simulink models, targeted to publish/subscribe middleware and Real-Time Operating System (RTOS)/processor combinations.

Although some code can be generated by UML/SysML tools, the focus is on coding interfaces (.h files) and state machines (conditional or switch statements). Coding for real-time embedded control systems (the primary focus of Millennium's Model Based Development Center, MBDC) is best done using a graphical programming language, such as Simulink or SystemBuild using an associated code generator. Wherein functional block selections invoke the complex underlying code required for closed loop dynamic simulation, and the code can be automatically parsed and altered to optimize it for fast execution and small memory footprint. To take advantage of the broad acceptance and usage of UML/SysML compliant COTS software tools, the MBDC has developed tools for directly importing UML/SysML interface definitions into a Simulink Model on the front end of a project wherein functional allocations and architectures have been implemented in UML or SysML for distribution across teams. Graphical programming languages have been used to create GNC and command/control code for several space flight and launch vehicles such as MSTI, DC-X, XSS10 and XSS11, and for many manned aircraft such as F-18 E/F, F-16 block D, F-35 and all aircraft developed by Airbus Industries. Recently, Simulink was used to code Guidance Navigation and Control (GNC), Electrical Power and Thermal Controllers for the LADEE Spacecraft, which was then integrated with C&DH and device driver code via the Core Flight System publish/subscribe middleware.

Code generated from the graphical model can be targeted to a specific Processor (eg. PowerPC, ARM, LEON), Operating System (eg. vxWorks, Linux, RTEMS) or a Publish/Subscribe Middleware layer (eg. CFS, DDS, Simitar discrete-event simulator). Targeting a Middleware gives the most portability, but has the most overhead. Targeting is accomplished by configuring a set of code-generation parameters in Simulink's Embedded Coder and a Template Language Compiler (TLC) file. Different coding languages have been generated in the past (such as ADA generated from SystemBuild using AutoCode for the MSTI spacecraft, or ADAcore's Qgen or SCADE's ADA code generator). However, currently, C and C++ code generation has the most support in terms of user base and technical support.

Automatically generated code can be less efficient, in either size or execution, than optimized hand-written code, but some code generators incorporate their own optimization features so as to generate code that approaches the "best" handwritten examples. Faster processors available starting in the early 1990's made code-generation targeted directly to the processor possible for missions such as that of the MSTI spacecraft. As higher power rad-hard processors, such as the Rad750, became available in the late 1990's, targeting to an Operating System became feasible. The LADEE mission targeted generated code to the Core Flight System (CFS) Publish/Subscribe Framework Middleware developed by Goddard Space Flight Center (now open-source), and CFS ran on top of the VxWorks real-time OS running on a RAD750.

All of the subsystems in a Graphical Model can be targeted to one monolithic function that is integrated with the rest of the system (device interfaces, command/telemetry functions), or separate subsystems can be targeted to separate tasks or threads that communicate with each other through some means such as shared memory, or an RTOS message queue or messages over a Pub/Sub middleware. Generating code from separate Simulink subsystems as CFS tasks allows the possibility of running them at different rates as well as managing updates to them separately.

LADEE targeted selected Simulink Top-Level Atomic Subsystems to be generated as loosely coupled CFS Applications. The interfaces (including rate-transitions) were coded via the code-generation TLC template as interactions with the CFS Pub/Sub Middleware. Simulink Inports were code-generated as CFS messages that were subscribed to. Simulink Outports were auto-coded as messages that were published. Scheduling of execution was accomplished via message reception from hardware that propagated messages through the system. The steps required to generate code from Simulink targeted to the Core Flight Executive (cFE) components of CFS are listed in Figure 16.

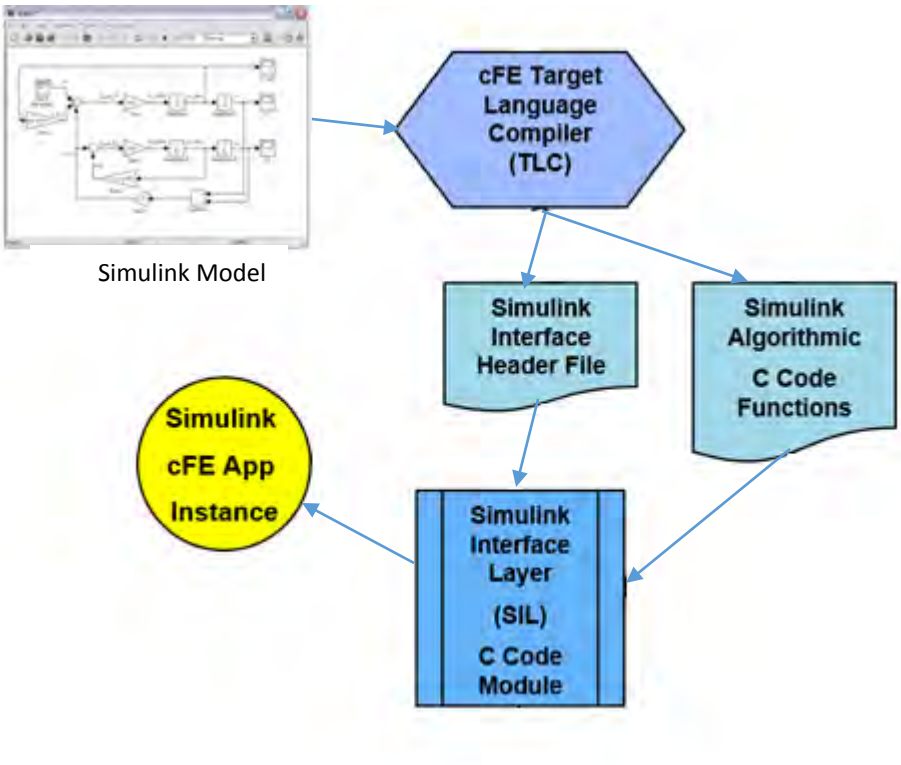

Figure 16: Generated code targeted to cFS/cFE middleware.

For the LADEE Processor in the Loop (PIL) testing, Flight Software Subsystems were targeted to a flight-representative processor and Simulation derived from the Workstation simulation described in section 4, consisting of sensors, actuators and dynamics subsystems targeted to CFE and a simulation processor. Data between the FSW and Sim processors was transported in various levels of fidelity as shown in Figure 17. Note that the exact same Simulink Models were code-generated into an evolving mixture of hardware configurations[13].

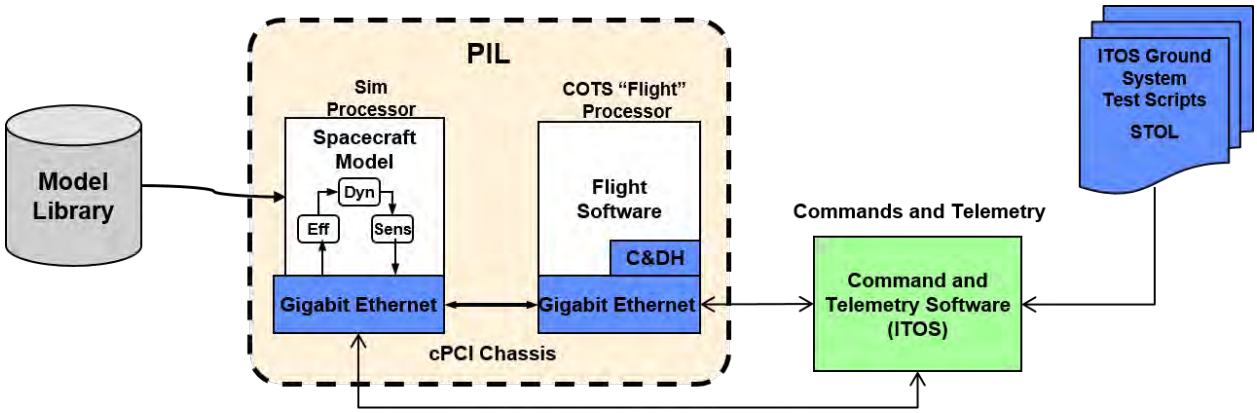

Figure 17: Processor In the Loop archetecture.

Another project developing space interceptor real-time HIL simulations for MDA has targeted Simulink models to the C++ interface of the SIMITAR discrete-event simulation framework. Changing the target of the control portion of the models from SIMITAR to cFE has been discussed. This can be accomplished with little or no changes to the Graphical Models. The Graphical Programming language provides the ultimate portability, since it does not rely on any particular target processor, language, Operating System or Middleware.

The other aspect of code generation is a mechanism that can create interface related code from a single data source in order to avoid errors caused by redundant data sources that need to be manually kept in sync. A simple example of an interface that lends itself to this mechanism is the Space-to-Ground interface between a vehicle and a remote ground system. Interfaces between on-board software modules also benefit from automatic interface generation. The single data source can be implemented in spreadsheets, a UML/SysMl model or a Relational Database Management System (RDBMS). The cleanest approach is an RDBMS with a clean schema that holds all interface related information along with tools that extract information from the RDBMS to generate documentation and code.

VALIDATING SIMULATION MODELS FROM TEST DATA

Since the MBD process creates a system design that reflects the behavior and fidelity of its underlying simulation, realization of the design as hardware and embedded software must also be tested in "real world" environments. In addition to verifying subsystem performance and providing a basis for accepting delivery of subsystems, data from subsystem hardware tests must be applied to support evolution of simulation fidelity as well as embedded software robustness and reliability. In MBD, testing is accomplished via simulations and PIL/HIL test systems initially, and then augmented by hardware testing to obtain excitation-driven data that enhances simulation model fidelity and prediction accuracy.

Sensors & Actuators

Sensors, actuators and other devices can be initially modelled in a Graphical Programming language 3 or 6 DOF dynamics simulation and specifications provided from the device manufacturer. A manufacture, such as Blue Canyon Technologies, may even provide a Matlab model of the device. These models can be used in closed loop simulations to develop initial control algorithms. As early as possible, when actual devices are acquired, they can be characterized in a suite of isolated real-life tests. The results of these characterizations can be then integrated back into the Graphical simulation. The Control Algorithms can then be modified as needed to match the higher fidelity

simulation of the sensors and actuators. This lowers the risk of encountering discrepancies when the actual devices are integrated into the system.

Complete systems

In some systems, such as a launch vehicle, test flights of sensors and actuators are used to characterize the sensors and actuators. If state estimation software, such as a Kalman Filter, is required to derive the most accurate state from a suite of sensors, some of which are expected to provide inconsistent results, the state estimation software may also needed to be characterized in an integrated test, such as a test flight. The NASA AVA program is proceeding in such a manner in its efforts to develop a more affordable avionics package that can be provided to small companies trying to develop game-breaking rocket technologies.

DEPLOYMENT AND OPERATION

One of the advantages of creating an integrated systems model, is the creation of a system simulator that can be used in the operational environment. In terms of spacecraft, the same system model that was created to develop and test the flight software can be use by the operations team to

1. Development and testing of spacecraft command scripts using Hardware-in-the-Loop (HIL) simulators before and during the mission
2. Operator training during mission simulations and readiness testing using the HIL simulator before the start of the mission
3. Verification of all tactical command sequence files uploaded using WSIM and PIL simulators

As an example, the model-based systems simulator developed by the LADEE flight software team was used before and during the operational phase of the mission. In order to train console operators a series of Simulations and Operational Readiness Testing (ORT) scenarios utilized the HIL simulator in or to provide a flight like training environment. The ORTs were conducted in real-time with operators on console 24 hours a day for 3-5 days depending on the mission phase being simulated.

A feature of simulator that was critical for operator training was the ability to inject faults into the system. During the flight software development the ability to inject faults was created to ensure the software satisfied requirements on fault tolerance. The mission operations test conductor was able to inject faults using the same feature in order to test console operators response faults. The test conductor has the ability to initialize the start of the simulation with faults (example: scale factor on thrust out of main axial thruster) or inject faults while the simulation is running (example: over-current temperature sensor failure).

The LADEE spacecraft simulator proved to be especially valuable during operations when debugging unexpected behavior of the star-tracker as shown in Figure 18. The guidance, navigation and control (GN&C) state estimator started showing performance errors shortly after activation, which triggered the on-board fault management to command the spacecraft into Safe Mode. The performance errors were the result of delayed reporting from the star tracker when a bright object (such as the Moon) entered the star tracker's field of view. The GN&C team was able to mimic the flight behavior of the star tracker using the simulator, develop and verify a patch for the on-board flight software, and enable the mission to successfully continue.

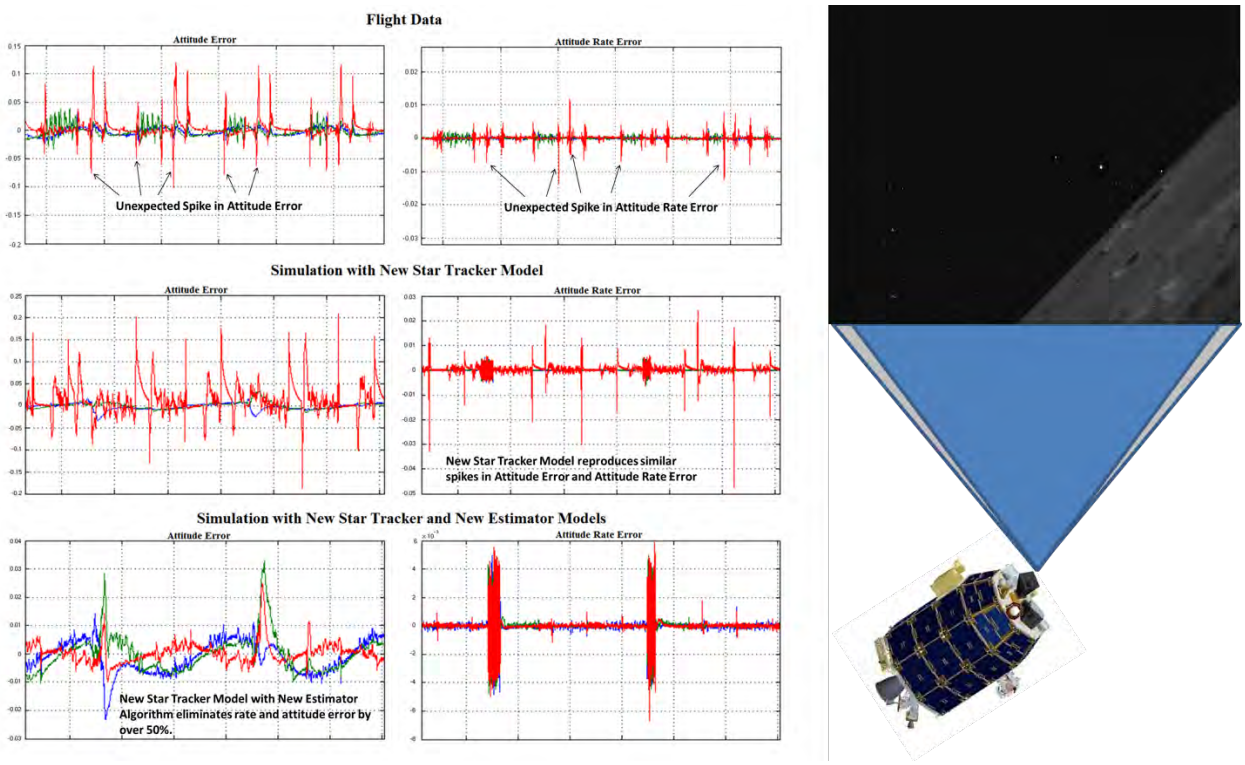

Figure 18: Using model based simulator to debug and fix flight anomaly.

SUMMARY

Model Based Systems Engineering (MBSE) and Model Based Development (MBD) are shown to be synomomous in describing simulation-driven, simulation-centric approach to defining, designing, implementing and deploying systems, wherein the Systems Engineering Process defined by DoD acquisition requirements is defined to encompass the entire End-To-End system acquisition cycle. This means that Systems Engineering encompasses engineering design, system realization and deployment into operation. Candidate suites of computer-aided design tools were presented that enable rapid, efficient and robust practice of systems engineering discipline across the entire end-to-end systems engineering cycle, and examples of their application to support development, deployment and operation of systems were presented. Details of block-diagram programming and code generation for system simulations and embedded software were presented. The ability to apply an evolving high-fidelity simulation model across all aspects of systems engineering, from requirements analysis to deployment and operation, is demonstrated and discussed. Deployment and use of the methods presented herein to develop complex systems can result in substantial enhancements in system robustness and reliability, ultimately increasing mission assurance and satisfying customer needs at reduced life cycle cost. The systems robustness benefits follow from the extensive simulatioin-driven testing enabled by a Model-Based approach across all aspects systems engineering from functional analysis through testing of physical systems

[1] M. Whalen, D. Cofer, S. Miller, B. H. Krogh and W. Storm, "Integration of Formal Analysis into Model-Based Development Process," in *Proceedings of the 12th International Conference on Formal Methods for Industrial Critical Systems.*, Berlin, 2008.

[2] M. Broy, S. Kirstan, H. Krcmar and S. Bernhard, "What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?," 2012.

[3] M. Ahmadian, Z. Nazari, N. Nakhaee and Z. Kostic, "Model Based Design and SDR (Software Defined Radio)," London, 2005.

[4] S. J. Toper and N. C. Horner, "Model-Based Systems Engineering in Support of Complex Systems Development," vol. 32, no. 1, 2013.

[5] M. Eigner, C. Muggeo, T. Dickopf and K. G. Faibt, "An approach for a Model Based Development Process of Cybertronic Systems," 2014.

[6] "Defense Acquisition Guidebook. Chapter 4 - Systems Engineering," [Online]. Available: https://dag.dau.mil/Pages/Default.aspx.

[7] "Systems Engineering Fundamentals", Supplementary text prepared by the Defense Acquisition University Press, Ft. Belvoir, VA 22060-5565, Dated Jan 1, 2001

[8] http://www.incose.org/AboutSE/WhatIsSE

[9] L. von Bertalanffy, General System Theory: Foundations, Development, Applications, New York: Braziller, 1968.

[10] K. Boulding, "General Systems Theory - Skeleton of Science," vol. 2, no. 3, 1956.

[11] N. Benz, D. Viazzo and K. Gundy-Burlet, "Multi-purpose Spacecraft Simulator for LADEE," Big Sky, 2015.

[12] K. Gundy-Burlet, "Validation and Verification of LADEE Models and Software," Pasadena, 2014.

[13] D. Forman, C. Pires and S. Christa, "Mix'n'Match Device I/O Transports for Evolving Closed-Loop FSW Testbed Fidelity," 2011.