Model-Based Verification and Validation of the SMAP Uplink Processes

M. Omair Khan
Jet Propulsion Laboratory,
California Institute of Technology,
4800 Oak Grove Dr.
Pasadena, CA 91109
818-354-3469
Mohammed.O.Khan@jpl.nasa.gov

Gregory F. Dubos
Jet Propulsion Laboratory,
California Institute of Technology,
4800 Oak Grove Dr.
Pasadena, CA 91109
818-354-0318
Gregory.F.Dubos@jpl.nasa.gov

Joseph Tirona
Jet Propulsion Laboratory,
California Institute of Technology,
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-8203
Joseph.F.Tirona@jpl.nasa.gov

Shaun Standley
Jet Propulsion Laboratory,
California Institute of Technology,
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-0936
Shaun.P.Standley@jpl.nasa.gov

Abstract— **Model-Based Systems Engineering (MBSE) is being used increasingly within the spacecraft design community because of its benefits when compared to document-based approaches. As the complexity of projects expands dramatically with continually increasing computational power and technology infusion, the time and effort needed for verification and validation (V&V) increases geometrically. Using simulation to perform design validation with system-level models earlier in the life cycle stands to bridge the gap between design of the system (based on system-level requirements) and verifying those requirements/validating the system as a whole.**

This case study stands as an example of how a project can validate a system-level design earlier in the project life cycle than traditional V&V processes by using simulation on a system model. Specifically, this paper describes how simulation was added to a system model of the Soil Moisture Active-Passive (SMAP) mission's uplink process.

Also discussed are the advantages and disadvantages of the methods employed and the lessons learned; which are intended to benefit future model-based and simulation-based V&V development efforts.

TABLE OF CONTENTS

1. INTRODUCTION 1
2. CASE STUDY / THESIS STATEMENT 2
3. DESCRIPTION OF THE UPLINK PROCESS 2
4. DETAILED DESCRIPTION OF MODEL (IMPLEMENTATION)..... 3
5. LESSONS LEARNED 7
6. SUMMARY..... 8
ACKNOWLEDGEMENTS..... 8
REFERENCES..... 8
BIOGRAPHIES..... 9

1. INTRODUCTION

System verification and validation at the Jet Propulsion Laboratory (JPL) is performed on the flight system during assembly, test, and launch operations (ATLO) on flight system testbeds with the associated simulation and support equipment (SSE), and on software simulators and other models of the system. For many years, we have been following the axioms that test is the preferred method of verification, and that we must test in the way that we intend to fly, and then fly the mission in the way that we have tested. These are worthy ideals to aspire to, but are increasingly difficult to meet in a time of ever more demanding missions, rapidly expanding processor capability, increasing software complexity, and decreasing resources for testing. For example, lightweight flexible structures interacting with system actuators, systems in planetary environments, and entry, descent, and landing scenarios are near impossible to test thoroughly at the system level.

Spacecraft generally spend a year or two in system integration, and test time with the flight system is so limited that it is difficult to perform more than a handful of nominal and faulted mission cases. System testbeds are available earlier and for longer, but their behavior cannot be compared to and certified against the flight system until the project is well into system validation testing (the right hand side of the Project Development V model [1]), where there is a large schedule and resource impact to mitigate problems that may arise.

Early use of highly capable, high fidelity software simulators, such as Work Station Test Set (WSTS) [2], are a major mitigation to the V&V problem, but adding ever more test resources to verify incomplete requirement sets for ever more complex spacecraft in ever more complex environments is not enough. We would like to do a better job of articulating the architecture of the system earlier in the project lifecycle, as well as validating that it can do the job before ever committing to hardware or software (like WSTS), and relying more on system models for verification and validation once the system is built. The Model-Based System Engineering approach of 'build the model, verify the model, build the system to realize the model, validate the system against the model' offers a way of getting a higher quality, lower cost result in the development of complex space systems. This study explored the utility of simulations generated directly from SysML models in validating operations concepts and scenarios early in the project lifecycle.

As part of the JPL Integrated Model-Centric Engineering and model-based V&V framework, we are exploring ways of integrating more model-based techniques in our systems

engineering work [4, 5], in particular performing system validation earlier in the lifecycle. One such method is to build a project model (comprised of a ground system, flight system, and associated subsystems), using the Systems Modeling Language (SysML), to a sufficient level of detail to validate operations scenarios and run early versions of operational readiness tests against the model. Advantages of such a model-based technique is not only the possibility of running simulations generated directly from the same models that express the architecture and design, but also the possibility of validating such models using formal methods [8] to obtain a far more detailed assurance that the design was valid for the expected range of conditions, and under all combinations of input parameters.

We are also intrigued by the possibilities of model re-use between different flight projects. Although initial model development is a significant task, the value of the effort will most likely be realized over several subsequent project development cycles. For spacecraft built using subsystem designs and architectural patterns that endure for several missions, major elements of the behavior models are also reusable. Although we do not yet have data on the effort it takes to change models to accommodate modifications in system design from mission to mission, it is clear that models for widely used standards (such as CCSDS) and common elements of missions' design (such as JPL's Multi-Mission System Architecture Platform, MSAP) will produce re-usable elements.

2. CASE STUDY / THESIS STATEMENT

This work involved creating a static model of the uplink system for the Soil Moisture Active and Passive (SMAP) spacecraft [3] using SysML, and generating a simulation directly from the model. The techniques and similar tools have been used before to model and simulate components of the spacecraft's avionics subsystem [4]. A similar methodology was used to model the uplink system, using newer tools and going into greater levels of detail. The intent was to build a functioning uplink simulation that could initially be used to validate the structure of the commands in successive deliveries of the SMAP spacecraft command database, and subsequently be integrated into a more generalized project model/simulation to account for spacecraft uplink processes.

Taking the elements of the command database and 'transmitting' them from a rudimentary ground system model, having the spacecraft model 'receive' and parse the SMAP Consultative Committee for Space Data Structure (CCSDS) [6] command structure correctly, routing the commands to the correct sequence engines then having them 'execute', and having corrupted commands be rejected by the modeled avionics with the correct error response would validate the uplink component of a later project-level model. In the current work, the CCSDS reception, parsing, and error detection aspects of the uplink process were validated via a system model-based simulation approach.

It was also intended that the static SysML model of the uplink system would be the subject of a model checking exercise – translating the SysML model to PROMELA [9], expressing the constraints and requirements on the system in computation tree logic, and proving the model met the requirements of the system; but regrettably this had to be left as a future task. Overall, this case study was an important step on our road to expressing architecture and design in models, validating those models early in the lifecycle with directly derived simulations and formal methods, building systems in accordance with the models, and performing the bulk of our V&V directly on the models. The intent of such an approach is to save ATLO testing time and resources for verifying hardware workmanship and performing a limited set of tests designated for model validation.

3. DESCRIPTION OF THE UPLINK PROCESS

The uplink process includes: the generation of commands, the radiation of these commands to the spacecraft, and their interpretation and execution by the flight system avionics.

First, the Mission Operations System (MOS) personnel input commands (like "flip a relay" or "cool boot the system") from a mission-specific Command Dictionary and use the Ground Data System (GDS) to transmit them to the spacecraft. These commands are formatted in a way that is consistent with the principles and recommendations formulated by the international Consultative Committee for Space Data Systems (CCSDS).

Each actual command only accounts for a small fraction of the contents of the final uplink message that gets transmitted to the spacecraft. Additional data is prepended and appended to the command in order to: ensure proper delineation of different units of data, help characterize the contents of the message and ensure space-link security.

Tele-command transfer frames are the basic data units that allow the transmission of user data to the spacecraft through the space data link [6]. Following the CCSDS recommendations, each transfer frame is encoded in elementary units called codeblocks that contain spacecraft command instructions, as well as additional bits used for error detection and correction. This provides protection against errors and corruptions that may occur over the space link. Multiple series of codeblocks are then encapsulated in a Communications Link Transmission Unit (CLTU), which provides a method for synchronizing the bit streams during reception [7]. Figure 1 illustrates a possible CCSDS structure of the uplink data, with each CLTU being delimited on both ends by fixed bit sequences called 'start sequence'(s) and 'tail sequence'(s). Finally, a command load (or command file) is defined as a contiguous bit stream containing several CLTUs. It starts with an 'acquisition sequence' containing various bits that enables the antenna in the spacecraft's telecom subsystem to acquire bit synchronization. It ends with an 'idle sequence' containing various bits that ensures that the antenna can maintain bit synchronization when no CLTUs are being transmitted.

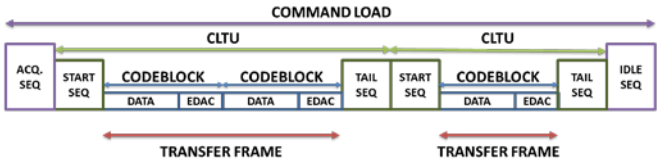

Figure 1 – Possible CCSDS structure of uplink data (Adapted from [6])

After the uplink message is received by the spacecraft's radio, it is directed via transponder to the Avionics (AVS) subsystem, namely the Telecom Interface card (TIF). The TIF is a Field Programmable Gate Array (FPGA) designed to: process various types of commands (by analyzing their CCSDS structure) and distribute them to different parts of the AVS subsystem. This process is necessary because some commands/files (e.g. direct hardware commands) need flight software (FSW) to be running concurrently in order to be executed while others do not. The TIF automatically executes commands that do not require FSW processing. The Spacecraft Flight Computer (SFC) processes all other commands and processes the data based on instructions from the ground. It records Event Verification Records (EVRs) at certain times when off-nominal or benchmark events occur. Figure 2 shows a high-level block definition diagram (BDD) of parts of the SMAP system that were modeled.

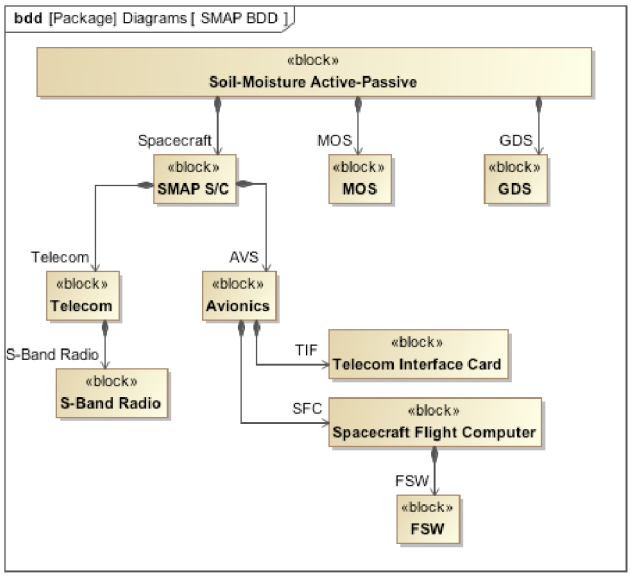

Figure 2 - SMAP High-Level System BDD

Figure 3 provides a "big picture" view of the SMAP uplink process, using the model elements described in Figure 2. This Internal Block Diagram (IBD) shows the direction of the data flow across the various subsystems.

4. DETAILED DESCRIPTION OF MODEL (IMPLEMENTATION)

Approach

The primary goal behind creating the model-based simulation was to determine the validation of the uplink system design. In order to do this, it was critical to set up a good strategy for interpreting how the simulation would provide design validation details. This was achieved by approaching subsections of the uplink process as 'black boxes' that were governed by design documents; upon integration of these 'black boxes' in the simulation, we could validate the design.

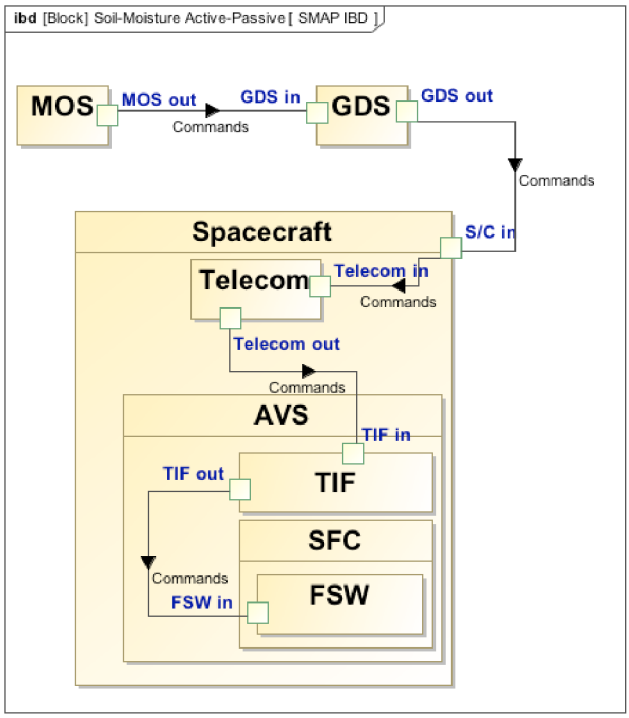

Figure 3 – Internal Block Diagram (IBD) for the general SMAP Uplink Process

The uplink process was split among the MOS/GDS, TIF, and FSW procedures. Each team member designed his aspect of the model per the corresponding design document and pre-determined input/output interfaces. In this way, the uplink simulation was built up as a series of functional "black boxes". Upon integration of the system, we found that the logic within each "black box" worked together to fully process nominal signals and catch faulty inputs and outputs. Due to this manner of building the simulation based upon an agreed design description, we were able to validate the design by testing the simulation against nominal and off-nominal inputs. The uplink design is based upon high heritage CSSDS formatting and processing, so it was no surprise that the design was valid; but of more importance is demonstrating this process of design validation by means of a system model-based simulation.

Methodology

Capturing the complex interactions and behaviors of the uplink process required the formulation of patterns to properly interpret aspects of the model in the simulation environment. The basic premise of the system model-to-simulation conversion methodology was based on treating state machine and activity diagrams as the visual representation of a traditional coding script. Code was interlaced into each activity node (e.g. action), so that the simulation engine executes the code as it comes upon each node.

The simulation environment can be thought of as a traditional script that is distributed among the various entities and diagrams in a system model. Although it can be quite different from traditional script writing at first glance, the methodology works just like a very large code with lots of function calls to hold it all together. The job of the simulation engine is to interpret the paths described in the various state machine (STM) and activity diagrams and walk through the logic in these diagrams while executing the code within each activity node along the way. Most SysML realization software packages have their own simulation plug-ins and work in a similar manner as described above.

We implemented our simulation engine per the methodology shown in Figure 4. Values properties on blocks are used a means for reading and writing changing variables during the runtime environment. They serve as one of the primary means of maintaining persistence in the model as the activity nodes are run as a series of independent-runtime environments. STM and activity diagrams represent the code architecture dispersed among the activity nodes as operator/function calls, where SysML artifacts (e.g. decision nodes, merges, etc.) can be used as if statements and for loops. Specific to our implementation we used opaque behavior and opaque actions as a means to insert code into the simulation using either JPython or Javascript syntax. As the simulation activates each of these activity nodes, it executes the underlying code. The ports and connectors on IBDs serve as a method to pass information between the underlying behaviors on each block in the system model.

This methodology was primarily used as it follows the SysML patterns that the system model is based on. Additionally, it allows the user to visualize and quantize what the static system model is trying to represent. This is especially important when dealing with a very complicated process like uplink – in this case a simulation enables quantitative and qualitative understanding of a design.

Figure 4 - Model to Simulation Conversion Methodology

Modeling Entity	***Simulation Implementation***
Values Properties on Blocks	Global Variables
State Machine & Activity Diagrams	Code Architecture
Opaque Behaviors & Actions	Operators / Function Calls
Ports/Connectors on Internal Block Diagrams	Module / Library / Super Class Interactivity

Details of model

Selected details of the model are shown in the figures below. Figure 5 is analogous to Figure 3 above; except that it is the actual IBD used in the simulation. Aspects that make Figure 5 vary are behavior diagrams overlaid onto the structural diagrams, which allow the user to track things like higher and lower-level processes running simultaneously during runtime and multiplicities of objects reflecting redundancy in the system.

Figure 5 is the diagram intended for users to watch during live simulations, to see where the "command" is in the system at any given point in time. This is a useful view because when a problem is found with the command or processing routine, the simulation highlights which subsystem is currently active and what operation it is performing. This helps in validating the model against the SMAP system, and eventually with testing responses to new commands.

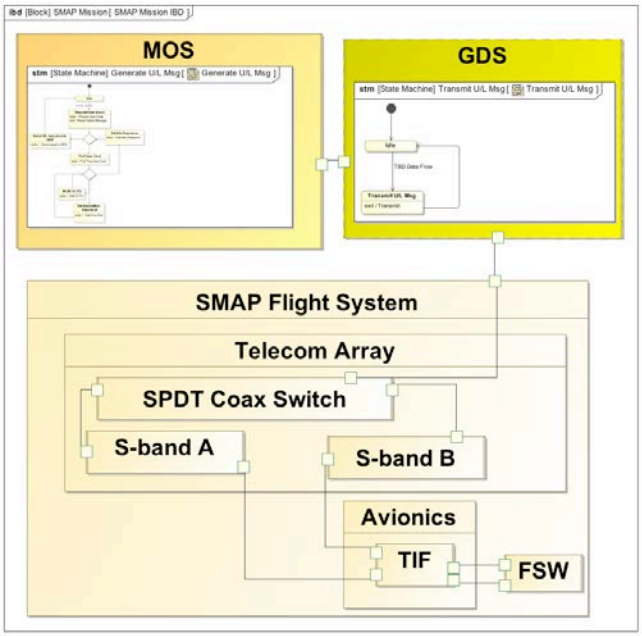

Figure 5 – Detailed SMAP Simulation IBD

Figure 6 is a state machine diagram (STM) controlling the command generation process. Starting with a request for the user to input a single (or list of) command(s), it invokes a series of other activity diagrams to encapsulate said commands into proper CCSDS format and sends them to the GDS for transfer to the flight system model.

Figure 7 shows the last activity invoked by the state machine represented in Figure 6. It takes the "U/L Msg" (Uplink Message) stored in a value property within the "MOS" section of the model and puts it on the "U/L Msg Out" port (seen as the only port leaving "MOS" in Figure 5) to simulate sending it to the GDS.

In Figure 7, the "Read Self" action is an artifact of the simulation, and is a visual equivalent to explicitly declaring a variable in "code". There are very few places where such "artifacts" are needed for simulation purposes. One advantage of simulation (discussed in further detail in the next section) is that the "overhead" involved with testing the system using MBV&V is on the order of variable declaration and dragging and dropping actions onto diagrams, which is far less work than developing high fidelity simulators such as WSTS [2] or a system test-bed.

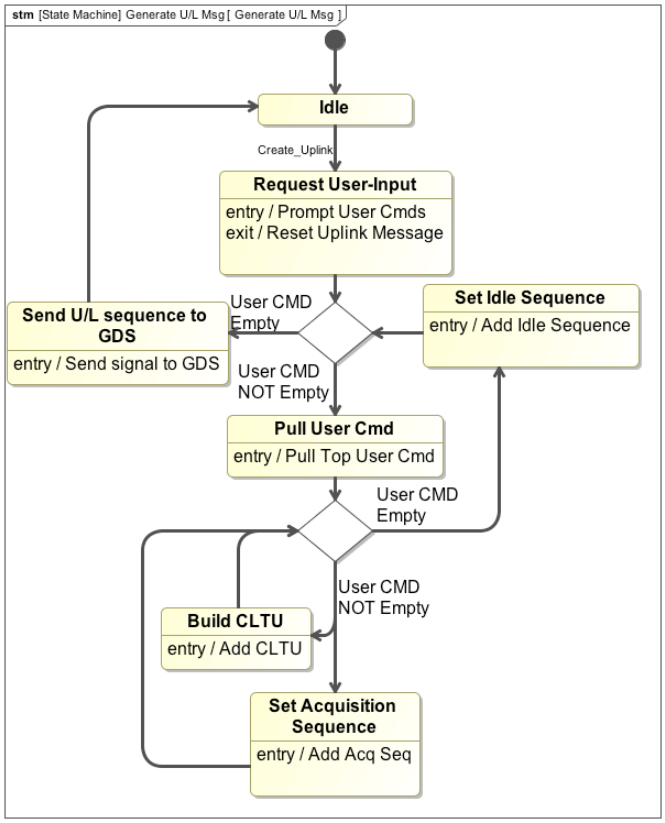

Figure 6 - Uplink Message Generation Process (Depicted as a STM)

Figure 8 shows the TIF's logic modeled as a STM. Because of the nature of FPGAs, its logic was not difficult to capture in the model since functionality is hard-wired into the board and was well defined in design documentation. Figure 8 also highlights how the STM was used to model a looping mechanism in the simulation (acting like a "for-loop"). The key advantage of using STMs over activity diagrams (with respect solely to the execution of the simulation engine) is that loops are permissible in STMs, while they are less straightforward in activity diagrams.

Figure 9 shows a diagram of the first activity invoked by the state machine represented in Figure 8 to run commands through the logic wired onto the TIF. While each action contains several embedded lines of code to manipulate the data flowing through the various subsystems during the simulation, the layout of the diagram and the titles of the actions suffice to convey the behaviors associated with each action. In summary, conveying this level of detail via the use of diagrams presents a major benefit: when a command is not executed successfully by FSW or by hardware on the spacecraft, there is a visual pointer that pinpoints the problem in the model/simulation.

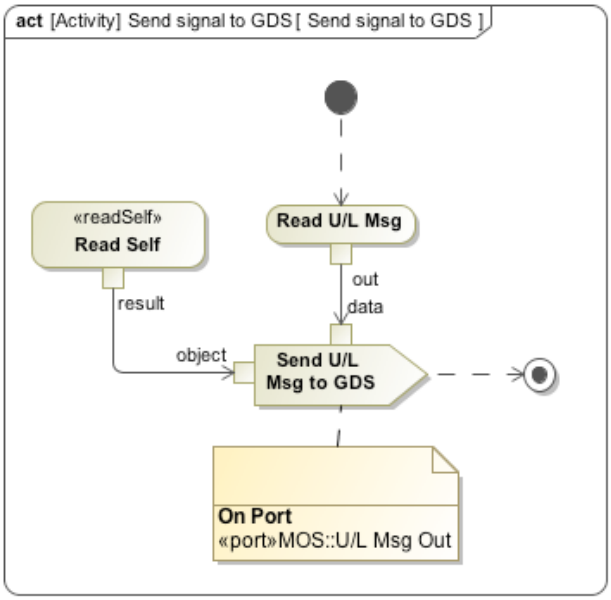

Figure 7 - Send Uplink Message Activity Diagram

Figure 11 is a testament to the levels of abstraction that can be built into the behavioral model of the system. As seen in Figure 8, one of the activities called by the "Process next codeblock" state is the activity called "Handle codeblock". Since this process is made up of several distinct processes, it was not recommended to place both of these complex processes onto the same diagram for sake of clarity.

Figure 10 takes this level of abstraction to even greater detail. When the action 'Place in the Buffer' in Figure 11 is executed, the simulation checks for uncorrectable errors per the logic and code in Figure 10 and displays them to the user.

Analogous to creating function calls in traditional programming, Figure 11 is meant for those viewers not directly concerned by the logic behind the FPGA and mostly interested in knowing whether or not "codeblocks were handled correctly". Part of the beauty of modeling tools with metadata and relationships lies in the unlimited number of the levels of abstraction that can be captured based on stakeholder needs and resources.

Another scenario in which abstraction is a key advantage is the case where a modeler inherits a large amount of code from a previous simulation. If this is a known quantity or if the project does not have the desire/resources to model certain parts of a system, these "black boxes" allow the project to focus only on modeling parts of the system that are of interest, while leveraging existing models. Because each behavior modeled has well-defined inputs and outputs (as well as logic that describes how data is being manipulated), it can be used in any model that adheres to those interface specifications.

Figure 12 represents an STM modeling the FSW portion of the spacecraft. Every time codeblocks are placed into the memory buffer, FSW is tasked with retrieving the data (from either buffer A or B). Upon gathering data from the memory buffer, FSW will reconstruct and check the data message for errors. If the data holds valid, FSW would then queue the command for execution, as seen in Figure 13.

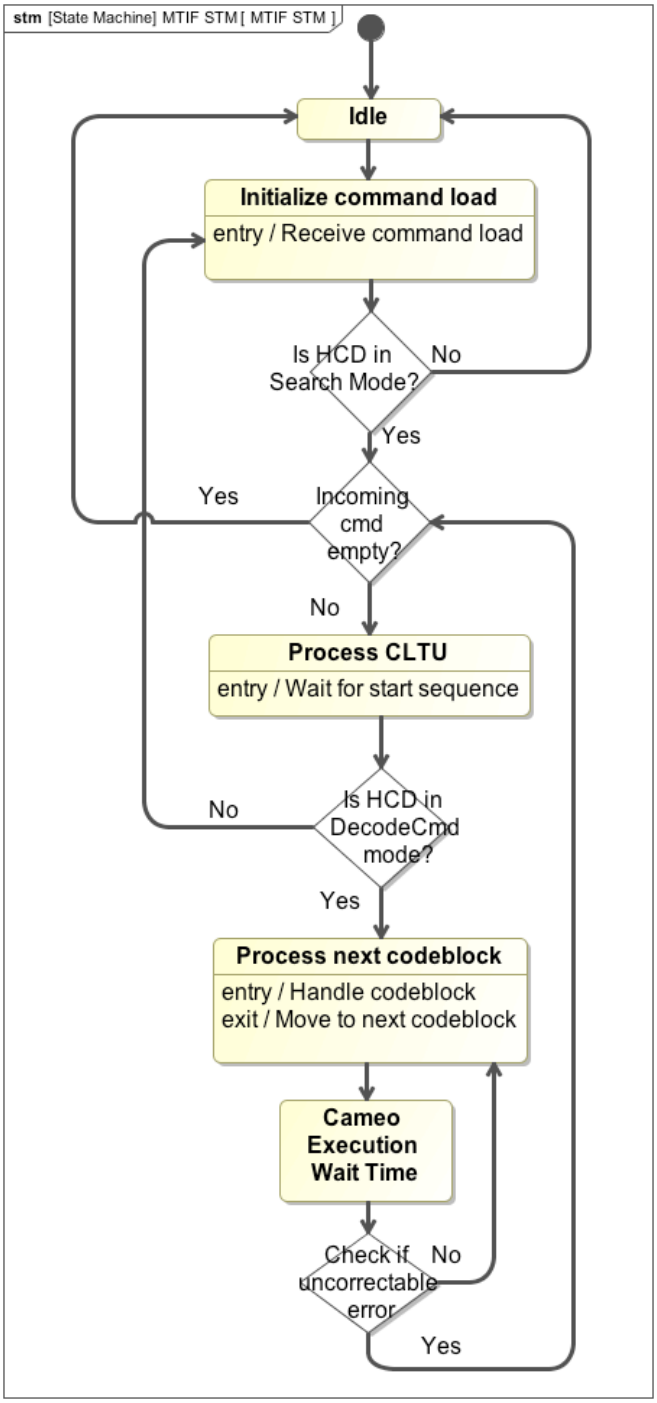

Figure 8 – TIF State Machine Diagram

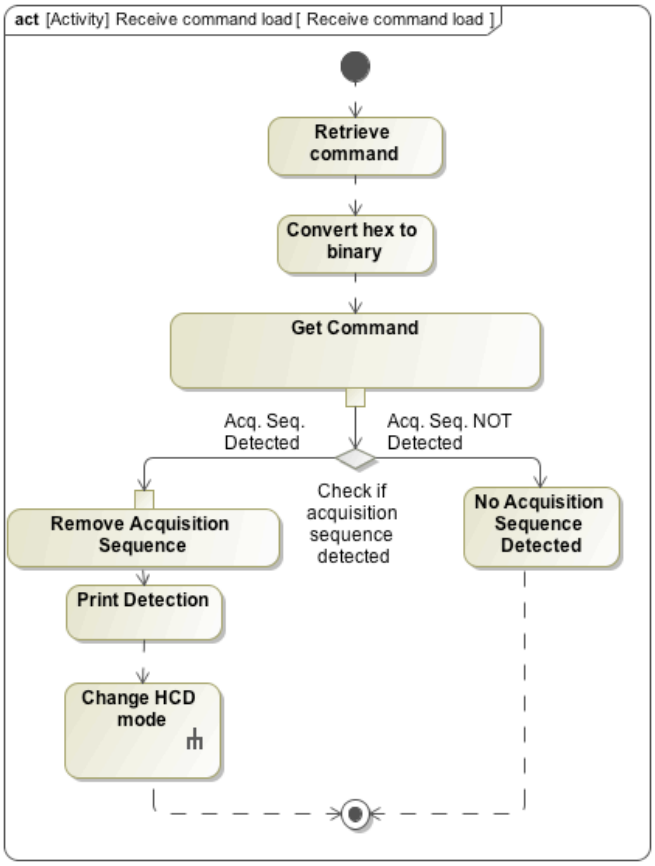

Figure 9 – TIF Command Reception Behavior

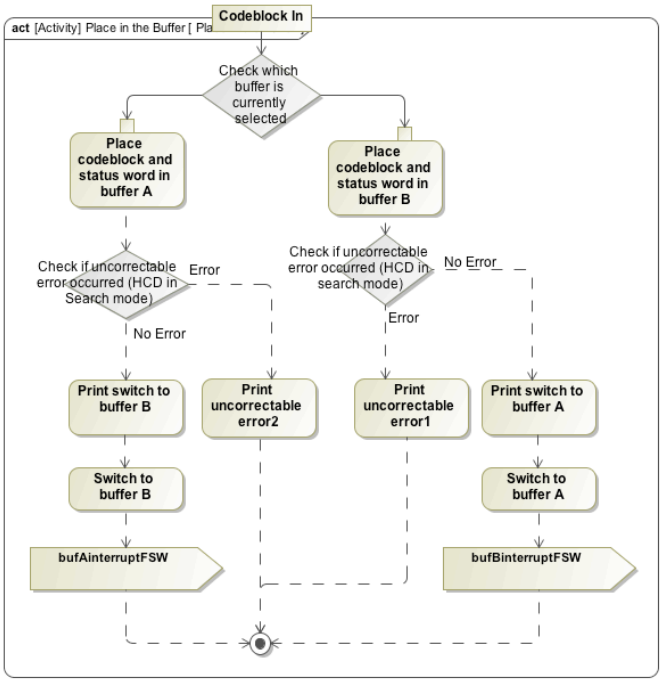

Figure 10 - Placing Codeblocks in the Buffer

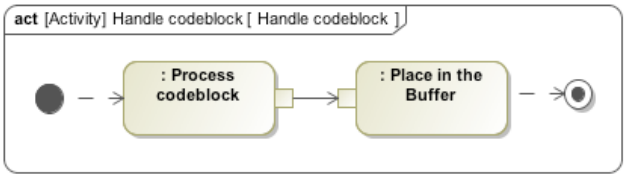

Figure 11 – TIF Codeblock Handling Behavior

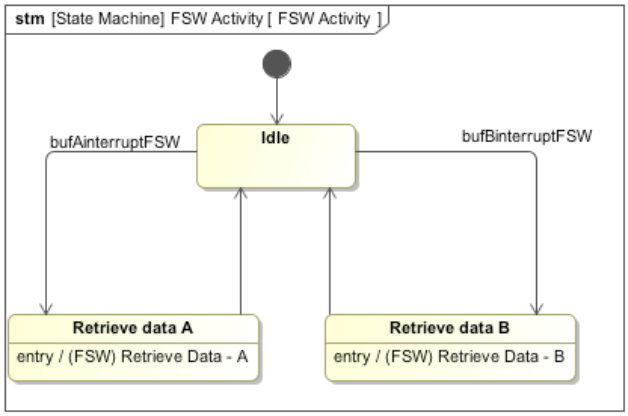

Figure 12 - FSW Data Retrieval Behavior

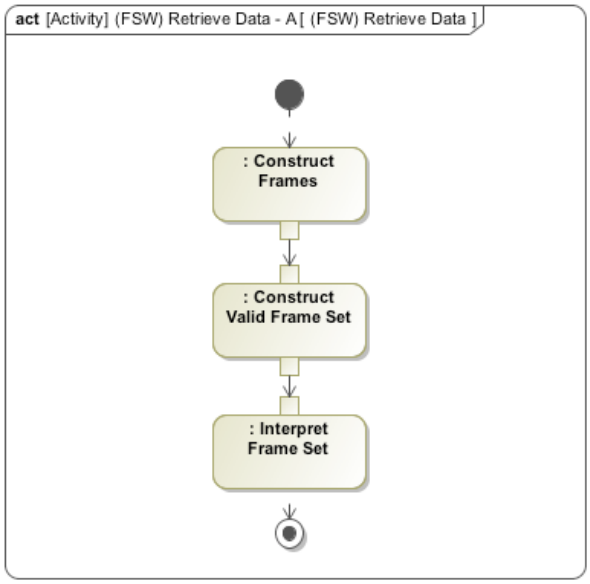

Figure 13 – FSW Frame Set Construction Activation Behavior

5. Lessons Learned

Compared to other simulation techniques relying exclusively on code, the use of a SysML-based simulation greatly enhanced the transparency and readability of the overall logical flow. Indeed, SysML enabled the organization of the entire simulation into distinct and well-identified behaviors containing individual pieces of executable code. As illustrated by several figures in the previous section, this offers any newcomer to the project a high-level view of the processes being simulated, regardless of their familiarity with the lower-level code. Transparency is also improved during the execution of the simulation, since behaviors become highlighted as they run, giving the end-user the chance to better comprehend the simulation runtime sequences. This feature is not only critical during debugging of the model, but also for qualitative validation (via visualizing the order that distinct operations are performed) of the overall uplink process. The simulation engine used to perform these simulations was completely off-the-shelf, and required only that the modelers learn how to use the program.

Another major advantage in running a simulation via a SysML model is the tight association between the structural model of the system (outlined in BDD's and IBD's) and the behavioral model of the system (outlining functionality and logic in STM's, activity diagrams, and parametric diagrams). This powerful combination of structure and behavior within the same environment provides an interesting means to reflect recent design changes and capture their impact on the behavior of the system. The logic of one or more modeled processes can be re-validated as new elements get added to the design (or modified).

During the development of the uplink model, various sections were modeled concurrently and independently, and unit tested before the end-to-end model was assembled. Development of the model therefore mimicked development of the actual hardware, in that lower level components are fabricated separately before being integrated as a functioning system. This process also provided model validation, guaranteeing the input and output data interfaces for each distinct subsystem model were consistent. The simulation provides an additional means to expose discrepancies between inputs and outputs at the interfaces between subsystems; as any mismatches will result in explicit anomalies when the simulation is run.

The decomposition of the uplink process into individual, logical pieces required the definition of various simulation artifacts to exchange information between these distinct pieces. While the main uplink message is directly modeled as a data flow that traverses the different activities or states, how this data is accessed and manipulated varies throughout the simulation. For example, the simulation takes advantage of both SysML "value properties" and "signals". Extra overhead is thus required to make sure that this data gets properly updated, in a way that resembles the use of global variables in traditional programming. However, this effort is minimal when compared to that effort involved in setting up a high fidelity software simulator such as WSTS. In addition, having a relatively accurate simulation much earlier in the life cycle, and with fewer resources, can prove particularly beneficial.

Finally, as with other simulation techniques, modeling parallel processes with SysML creates some challenges in terms of synchronization. Extra care must be taken to ensure that model elements trigger the appropriate responses at the right times (with respect to other elements in the simulation). Timing issues may have to be addressed, depending on the fidelity of the simulation engine being used and the number of desired concurrent operations. During the implementation of this work, it was identified that the simulation engine's speed of execution and the computing speed of the machine could impact the logic. This is a major limitation of the simulation

engine that was used that had to be circumvented by implementing artificial delays to ensure the proper sequencing of activities.

6. SUMMARY

Simulation was successfully added to the structural and behavioral models of the SMAP mission. Although some idiosyncrasies of the tooling needed to be worked out, the general overhead with creating a simulation from the static models was miniscule when compared to creating a high-fidelity software simulator like WSTS [2] or an avionics hardware testbed. The benefits of this SysML simulation allowed for verification of the SMAP mission's actual command dictionary per the logic outlined in design documents; thus successfully validating the system design with a small team and limited resources.

Unfortunately, this process was developed concurrently as hardware testbeds and WSTS [2] were being developed; so the full effect of this simulation was not realized early enough. But as future missions (and other complex systems in general) are proposed and funded, they can take advantage of this process to validate high-level functionality of their designs before copious amounts of money are spent validating the design at lower levels.

To date, the model-based V&V [MBV&V] paradigm at JPL has focused around system model-based simulations [4] and system test diagrams [5], but this does not fully satisfy the goals of MBV&V, which are to address all areas of V&V when working with system models. In current studies, the applications of MBV&V are being extended to demonstrate interactions between a SysML system model and a high-fidelity avionics simulator, such as WSTS [2]. Additionally, work is also being done to demonstrate interaction between a SysML system model and an avionics testbed. The intent of this current work is to extend the application of MBV&V to the implementation phase of the project development "V" [1].

ACKNOWLEDGEMENTS

The work described in this paper was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The authors would also like to thank Division 31 (SSCAE - Chi Lin) for supporting this work.

REFERENCES

[1] Figure 3-2 of NPR 7123.1A, reiterated in Figure2.1-1 of the SE Handbook

[2] Henriquez, Canham, Chang, McMahon; *Workstation-Based Avionics Simulator to Support Mars Science Laboratory Flight Software Development*, AIAA Modeling and Simulation Technologies Conference, 18-21 August 2008, Honolulu, HI.

[3] Official JPL site for the Soil Moisture Active and Passive (SMAP) mission http://smap.jpl.nasa.gov/

[4] Khan, Sievers, Standley, *Model-Based Verification and Validation of Spacecraft Avionics*, Infotech@Aerospace 2012 Conference, Santa Ana, CA, May 2012.

[5] Ingham, Day, Donahue, Kadesh, Kennedy, Khan, Post, Standley, *A Model-Based Approach to Engineering Behavior of Complex Aerospace Systems*, Infotech@Aerospace 2012 Conference, Santa Ana, CA, May 2012.

[6] Consultative Committee for Space Data Systems, *Telecommand Summary of Concept and Rationale*. CCSDS 200.0-G-6. Green Book. Issue 6. January 1987.

[7] Consultative Committee for Space Data Systems, *TC Synchronization and Channel Coding—Summary of Concept and Rationale.* CCSDS 230.1-G-1. Green Book. Issue 1. June 2006.

[8] *Verification and Validation in Systems Engineering – Assessing UML/SysML Design Models*, Debbabi, Hassaine, Jarraya, Soenu, Alawneh, Sprindger-Verlag 2010, ISBN 978-3-642-15228-3.

[9] *Model Checking with SPIN,* http://spinroot.com/

BIOGRAPHIES

Omair Khan *has been with JPL for more than 4 years. His work includes system-engineering support on future NASA astrophysics proposals and further developing the paradigm of system model-based V&V. He is currently supporting the DARPA Phoenix project, where he is seeking solutions to do system V&V using SysML models and high-fidelity simulators. He received a M.S. in Aerospace and Mechanical Engineering from the University of Southern California and a B.A. in Physics and Astrophysics from the University of California, Berkeley.*

Dr. Gregory Dubos *is a Systems Engineer in the Mission Systems Concept Section at the Jet Propulsion Laboratory (JPL). His current work at JPL focuses on the infusion of Model-Based Systems Engineering (MBSE) and simulation during mission formulation activities. For DARPA's Fractionated Spacecraft (F6) project, he developed discrete-event simulation models that capture technical risks such as launch failures, spacecraft failures, and orbital debris. He also regularly serves as the Risk and Programmatics Chair for Team X, JPL's concurrent engineering environment. Dr. Dubos received his B.S and M.S degrees in Aeronautics from SUPAERO, Toulouse, France, and his M.S and Ph.D. in Aerospace Engineering from the Georgia Institute of Technology, Atlanta, GA.*

Joseph Tirona *is a member of the System Verification and Validation Group at JPL. He has used MBSE to model Ground Data Systems on a number of flight projects, including the Mars Science Laboratory (MSL), Dawn, and the Orion Multi-Purpose Crew Vehicle Program. He is currently working as a Systems Engineer doing V&V on the Soil Moisture Active-Passive (SMAP) mission. He received a B.S. in Mechanical Engineering from California Polytechnic University, Pomona, where he is a graduate student.*

Shaun Standley *is the Group Supervisor for the System Verification and Validation Group at JPL. His current interest is using model-based techniques to improve the system verification and validation process. He has held a number of positions in V&V Engineering and spacecraft operations, including Project V&V Engineer for the Soil Moisture Active and Passive (SMAP) mission, End to End Information System engineer for the Kepler mission, Spacecraft Engineer for the Cassini orbiter, Systems Engineer for the Huygens Titan probe, and Systems Engineer for the Ulysses Solar orbiter. He received a B.Sc. (Hons) from Southampton University in the UK where he also did postgraduate work, and an MBA from Pepperdine University.*