Modeling in the Stateflow® Environment to Support Launch Vehicle Verification Testing for Mission and Fault Management Algorithms in the NASA Space Launch System[*]

Luis Trevino, Ph.D.[†],
Jacobs Technology
NASA Marshall Space Flight Center
Huntsville, Alabama 35812

Peter Berg[‡]
NASA Ames Research Center
Moffett Field, CA 94035

Dwight England[§]
NASA Marshall Space Flight Center
Huntsville, Alabama 35812

Stephen B. Johnson, Ph.D.[**],
Dependable System Technologies, LLC, and
University of Colorado, Colorado Springs
Colorado Springs, Colorado

Analysis methods and testing processes are essential activities in the engineering development and verification of the National Aeronautics and Space Administration's (NASA) new Space Launch System (SLS). Central to mission success is reliable verification of the Mission and Fault Management (M&FM) algorithms for the SLS launch vehicle (LV) flight software (FSW). This is particularly difficult because M&FM algorithms integrate and operate LV subsystems, which consist of diverse forms of hardware and software themselves, with equally diverse integration from the engineering disciplines of LV subsystems. M&FM operation of SLS requires a changing mix of LV automation. During pre-launch the LV is primarily operated by the Kennedy Space Center (KSC) Ground Systems Development and Operations (GSDO) organization with some LV automation of time-critical functions, and much more autonomous LV operations during ascent that have crucial interactions with the Orion crew capsule, its astronauts, and with mission controllers at the Johnson Space Center. M&FM algorithms must perform all nominal mission commanding via the flight computers to control LV states from pre-launch through disposal and also address off-nominal failure conditions by initiating autonomous or commanded aborts (crew capsule escape from the failing LV), redundancy management of failing subsystems and components, caution and warning notifications to crew and

[*] SPACE 2016, AIAA Space and Astronautics Forum
[†] Integrated Systems Health Management (ISHM) and Automation Branch, Spacecraft and Vehicle Systems Department, EV43
[‡] M&FM State Analysis Lead, Stinger Ghaffarian Technologies
[§] Chief, ISHM, Spacecraft and Vehicle Systems Department, EV43
[**] Analysis Lead, Mission and Fault Management, Spacecraft and Vehicle Systems Department, EV43

ground, and safing actions to reduce or prevent threats to ground systems and crew.

To address the criticality of the verification testing of these algorithms, and specifically the need for an early look at algorithm success criteria, the NASA M&FM team has utilized the Stateflow® environment[††] (SFE) to complement its existing Vehicle Management End-to-End Testbed (VMET) platform which also hosts vendor–supplied physics-based LV subsystem models. The human-derived M&FM algorithms are designed and vetted in cross-program Integrated Development Teams (IDT) composed of design and development disciplines such as Systems Engineering, FSW, Safety and Mission Assurance (S&MA) and major subsystems and vehicle elements such as Main Propulsion Systems (MPS), Boosters, Avionics, Guidance, Navigation, and Control (GN&C), Thrust Vector Control (TVC), Liquid Engines, Upper Stage Vehicle, Crew Vehicle, and the Astronaut Crew Office. The algorithms are designed using model-based engineering (MBE) methods from a hybrid of the Unified Modeling Language (UML) and Systems Modeling Language (SysML). The use of MBE was determined by the SLS program (SLSP) to be the most cost effective platform to allow digital simulations to, in some cases, take the place of very costly live hardware tests. Similary, these UML/SysML modeling languages were selected to bridge the gap between hardware design and certified FSW code as it would provide a logical view of the algorithms and therefore allow a more thorough review of software function by program engineers who are not experts in software programming. This approach was viewed to be not only a cost saver over time, but also vital in providing increased confidence of the design via extensive simulation testing. SFE methods are a natural fit to provide indepth analysis of the interactive behavior of these algorithms with the SLS LV subsystem models. For this, the M&FM algorithms and the SLS LV subsystem models are integrated using constructs provided by MATLAB®, which also enables modeling of the accompanying interfaces, providing greater flexibility for integrated testing and analysis. The SFE provides more rapid independent assessment of algorithm accuracy than standard testing and forecasts expected behavior in future VMET integrated testing activities. The combination of SFE and integrated testing provides a thorough independent assessment to buy-down risk in the formal FSW certification process.

In VMET, the M&FM algorithms are prototyped and implemented using the same C^{++} programming language and similar state machine architectural concepts used by the NASA FSW group. Due to the interactive complexity of the algorithms, VMET testing thus far has verified all the individual M&FM subsystem algorithms against select subsystem vendor models and is steadily progressing to assess the interactive behavior of these algorithms with the LV subsystems, as represented by subsystem models. The novel SFE application has proven to be useful for quick look analysis into early integrated system behavior and assessment of the M&FM algorithms with the modeled LV subsystems. This early MBE analysis generates vital insight into the integrated system behaviors, algorithm sensitivities, design issues, and has aided in the debugging of the M&FM algorithms well before full testing can begin in more expensive, higher fidelity

[††] Part of the suite of tools with Simulink by the MathWorks Inc., an American corporation specializing in mathematical computing software

environments such as VMET, FSW testing, and the Systems Integration Lab[‡‡] (SIL), therefore avoiding significant schedule delays or potential revisions to the software code late in the FSW verification process. The SFE has exhibited both expected and unexpected behaviors in nominal and off nominal test cases prior to full VMET testing. In many findings, these behavioral characteristics were used to correct the M&FM algorithms, enable better test coverage, and develop more effective test cases for each of the LV subsystems. This has improved the fidelity of testing and planning for the next generation of M&FM algorithms as the SLS program evolves from non-crewed to crewed flight, impacting subsystem configurations and the M&FM algorithms that control them. SFE analysis has improved robustness and reliability of the M&FM algorithms by revealing implementation errors and documentation inconsistencies. It is also improving planning efficiency for future VMET testing of the M&FM algorithms hosted in the LV flight computers, further reducing risk for the SLS launch infrastructure, the SLS LV, and most importantly the crew.

I. NASA Stateflow® Inception into Systems Engineering of Spacecraft Systems

Preventing errors from occurring in mission (including fault) management systems is the central theme of the M&FM test team working safety-critical system algorithms for FSW implementation for the SLS program (SLSP). The SLS is NASA's newest LV vehicle currently under development with the initial design configuration scheduled for launch in 2018. This configuration consists of the Core Stage (CS) liquid propulsion system and two Solid Rocket Boosters (SRBs), both derived from the Space Shuttle program, the Interim Cryogenic Propulsion Stage (ICPS), and the Orion/Multi-Purpose Crew Vehicle (MPCV) spacecraft. In LV systems the software implementation for fault management algorithms may entail up to 50% or more of the total system, meaning there is a considerably high probability that if systems software errors do occur, they can be attributed to or can affect fault management algorithms[1]. With current best practices, it is known from software engineering that 70% of errors are introduced during the requirements, system design, and software design phases, while 80% of errors are not discovered until integration and acceptance testing[1]. Furthermore, research shows that 20% of the errors are introduced during code development and unit testing[1]. In the requirements phase alone research examining NASA data on system errors reveals requirement-related error categories as omitted/missing requirements (33%), incorrect requirements (24%), incomplete requirements (21%), ambiguous requirements (6.3%), over specified requirements (6.1%), and inconsistent requirements (4.7%)[1]. These types of errors, known as architectural or design errors, are the ones most often implicated in actual accidents[1]. Because architectural errors can never be completely eliminated in complex systems, M&FM methods must also ensure that the failure effects of errors are adequately mitigated. In essence, the system portion responsible for reliable operations of the LV could in itself be unreliable. Systemic root causes stem from a variety of factors, including the operational environment, system components, subsystem interactions, human interpretation for implementation of requirements, redundancy levels of hardware and software, and more[2]. A major root cause is the challenges of testing the fault management portions of the overall LV subsystems. Since fault management software is only executed when a particular subsystem fails, and some failures occur only when the suspect subsystem is already dealing with a previous failure condition(s), some fault management software may not be fully tested for extreme or unforeseen conditions[1]. This, coupled with concurrent and/or time-delayed faults, only further compounds issues. There is a clear need for architectural models with well-defined semantics that support analysis of mission and safety-critical requirements with clear behavioral analysis. The intent of this paper is to introduce a cohesive approach to using Stateflow® with existing M&FM constructs to improve the overall quality and confidence of the M&FM algorithms for the SLSP.

NASA has embraced the SFE technology as part of the MBE design regime in various spacecraft applications in recent years as a tool to improve algorithm test coverage and therefore confidence in FSW. One notable application is the Lunar Atmosphere and Dust Environment Experiment (LADEE) spacecraft and its simulator[3] (Berg et al) which orbited around the Moon's equator studying the lunar exosphere and dust. Stateflow® methods have also been used to investigate commercial spacecraft applications for NASA and for automotive systems under the charter of the NASA Engineering and Safety Center[4] (Berg et al), notably the Toyota investigation of unintended acceleration. The SFE for the SLS work described in this paper leverages the Mathworks® Company products and its wide applications in various

[‡‡] Hardware in the loop facility hosting FSW and M&FM algorithms

industries. The Stateflow® toolbox in MATLAB® provides a rich set of graphical modeling constructs to permit users to quickly describe hierarchical and/or parallel complex systems and its simulation capability helping users visualize system behavior proving further levels of assurance at early development stages for a system[5]. This is particularly useful since parallelism can have two or more orthogonal states active at the same time requiring in depth analysis[6]. This coupled with the M&FM group's prior utility of UML-SysML for modeling M&FM behavior provides a rich set of methods for modeling and understanding requirements enabling the visualization of expected behavior. Stateflow® uses a variant of the popular concept of finite state machine notation in which a representation of an event-driven (reactive) system transitions from one state (mode) to another prescribed state, provided that the condition defining the change is true[6]. In MATLAB® Stateflow® states and transitions form the basic building blocks of a system, exemplified by Figure 1 below exhibiting the modeling of a component (Component_1) capable of transitioning between two states A and B[§§]. The conditions for transitioning between the two states are respectively shown as condition_2 and condition_1 and from the state flow lexicon viewed as Guard statements. Such conditions can be represented by an event, a command, or even a state from another component state machine. Further details of Stateflow® modeling is beyond the scope of this paper but presented further in prior work (Berg et al.)[7], with relevant terminology listed in the Appendix.

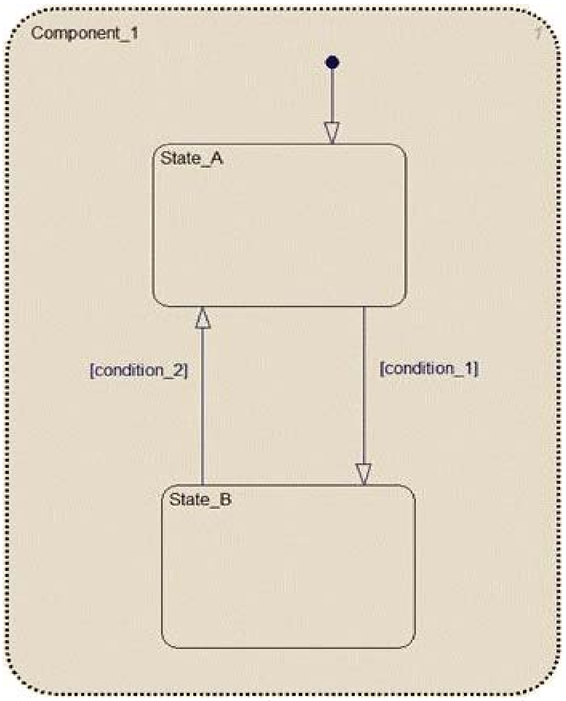
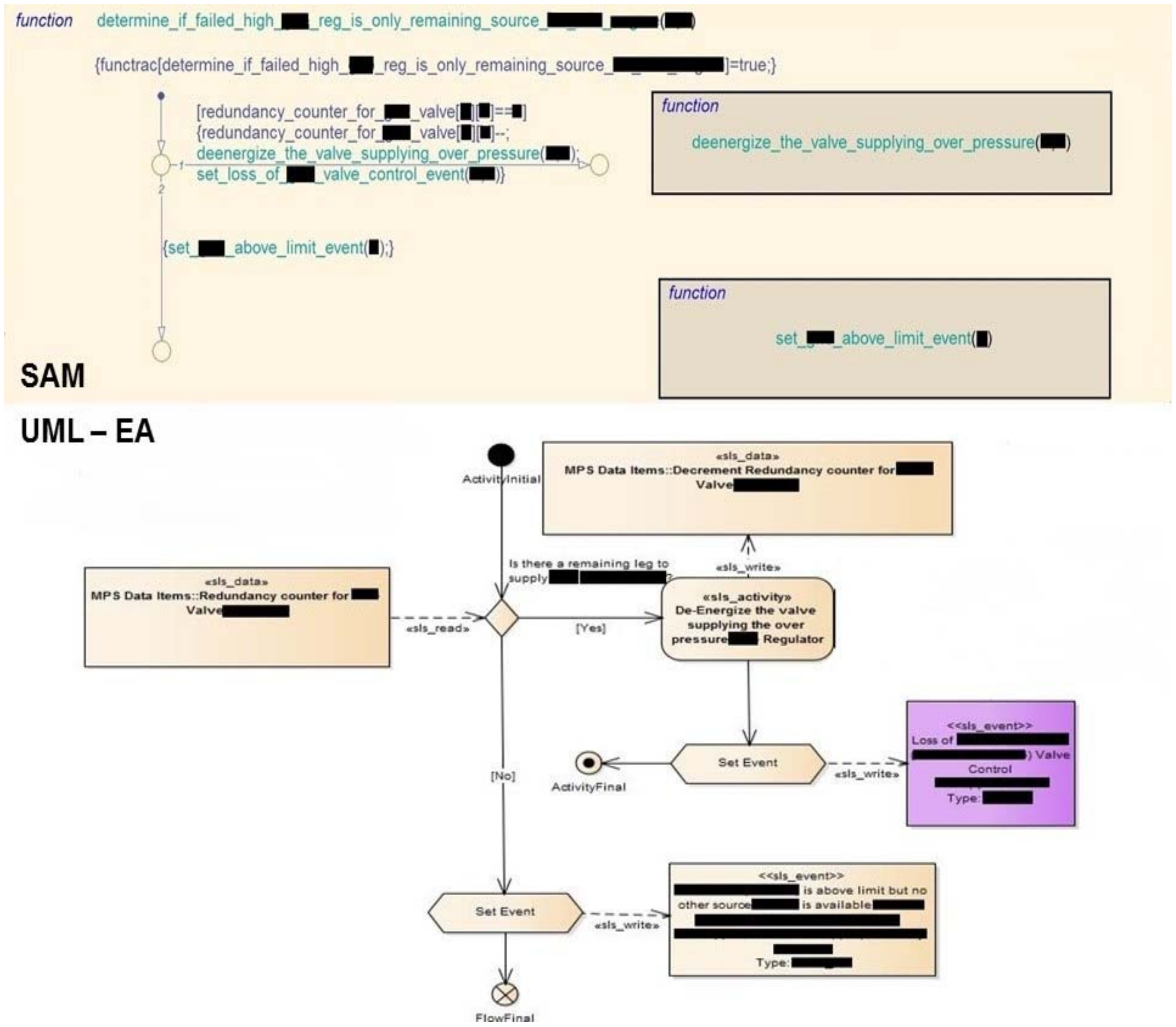


Figure 1. MATLAB® Basic State Machine [8]

For the SLSP, the SFE was used to model the avionics hardware and software (M&FM algorithm) components, presented further in the next section. The end product of this modeling effort became termed the State Analysis Model (SAM). The software components were modeled in the SAM using the M&FM algorithms which were developed using Enterprise Architect (EA). This modeling software utilizes the UML-SysML technology which is beyond the scope of this paper but is presented further in prior work[2] (Trevino et al). Figure 2 below exhibits a censored example of the two modeling technologies illustrating analogous representations of actions to be taken in the event of an over pressure detection in the helium system for the liquid propellant engines. For SLSP the benefits of utilizing UML-SysML were established in prior space programs (e.g., Ares I). Implementing SAM via the SFE provides a boon for further understanding the dynamic interdependent and interactive behavior across all the subsystem algorithms through early testing well before higher fidelity costly testing.

[§§] The underscore "_" notation indicates variables and states in MATLAB



function
determine_if_failed_high_[redacted]_reg_is_only_remaining_source_[redacted]([redacted])
{functrac[determine_if_failed_high_[redacted]_reg_is_only_remaining_source_[redacted]]=true;}
[redundancy_counter_for_[redacted]_valve[[redacted]][[redacted]]==[redacted]]
{redundancy_counter_for_[redacted]_valve[[redacted]][[redacted]]--;
deenergize_the_valve_supplying_over_pressure([redacted]);
set_loss_of_[redacted]_valve_control_event([redacted])}
1
2
{set_[redacted]_above_limit_event([redacted]);}
function
deenergize_the_valve_supplying_over_pressure([redacted])
function
set_[redacted]_above_limit_event([redacted])
SAM
UML – EA
ActivityInitial
«sls_data»
MPS Data Items::Decrement Redundancy counter for [redacted] Valve [redacted]
Is there a remaining leg to supply [redacted]?
«sls_write»
«sls_data»
MPS Data Items::Redundancy counter for [redacted] Valve [redacted]
«sls_read»
[Yes]
«sls_activity»
De-Energize the valve supplying the over pressure [redacted] Regulator
Set Event
ActivityFinal
«sls_write»
<<sls_event>>
Loss of [redacted]) Valve Control [redacted]
Type: [redacted]
[No]
Set Event
«sls_write»
<<sls_event>>
[redacted] is above limit but no other source [redacted] is available [redacted]
Type: [redacted]
FlowFinal

Figure 2. State Flow Modeling Representation (censored) Comparison with UML Modeling Representation for Gaseous Helium over Pressure Off-Nominal Condition for the MPS System

Justification for employing MATLAB® Stateflow® methods for the SLSP stems not only from the complexity of the required M&FM algorithms for ensuring reliability and assurance of the LV subsystems during nominal and off-nominal conditions, but from the added complexities of the supporting hardware infrastructure interacting with the algorithms. The computing platform for the SLSP is a triple redundant system interacting with other distributed networked systems such as the MPS, e.g., combined control system electronics (CCSE), cryogenic level sensor system (CLSS). The intent of this redundancy is to provide further reliability with replicated instances of the embedded software on different processors with communication over different network instances (i.e., 1553 buses, telemetry pathways). A change in the deployment configuration of the embedded software may lead to replication of software components' allocation to the same physical processor and to the elimination of physical redundancy[1]. For the SLS, the computing processor is based on the ARINC 653[***] partitioned runtime architecture and in this case can result in reduced reliability if the mapping of the embedded software to partitions and the binding of partitions to physical hardware are not performed consistently with the redundancy requirements for the system. Finally, any virtualization in the computing platform can lead to unplanned resource contention and performance that is slower than expected[1]. Further details on the computing platform is beyond the scope of this paper.

[***] Avionics standard for safe partitioned operating system developed by the Wind River Systems

It is important to note that state flow methods are not without their own issues. The state transition logic embedded in the state machines may make assumptions about the interaction of replicated and/or mirrored state machines by working with the same inputs exclusively, by comparing states periodically, or by observing each other's state behavior in order to detect anomalous behavior. The state logic may not accommodate failures in the application logic, in the underlying hardware models, or in any timing differences due to any asynchronous behavior from the modeled computing platform. Another concern is that the state transition logic may not fully model the communication of state versus state change, e.g., exchange of track information and track updates, where state change information assumes guaranteed and often-ordered delivery of information by the communication protocols and hardware. Also the communication of events by sampling state variables – the particular implementation, while maintaining a periodic task set, may not always guarantee observation of every event or queuing of events if arrival bursts exceed the processing rate due to any mismatch in any paradigms of guaranteed event processing and data sampling [1]. This issue, also called message storming is planned to be addressed in forward work by the state flow NASA team. State actions, condition actions, and transition actions provide the user flexibility on when actions take place in relation to state transitions. While combining the constructs with junctions and feedback loops can facilitate more complex logical implementations, mixing them can result in unintended consequences. Thus, careful design is warranted. Because of this potential for confusion and the degree of redundancy between options, Mealy standards specify that only condition actions are permitted[8]. The actual standards used for SLSP are covered in the next section. The complexity of these issues drives the need to employ a variety of methods for design and simulation of the M&FM algorithms, i.e., MBE design and VMET simulation platform where the algorithms are prototyped and unit tested using similar C^{++} constructs as that used by FSW[2]. Each approach makes certain assumptions about the content of the model and therefore embeds a-priori knowledge about how the model information is organized and implemented.

Despite such concerns, using state flow methods is viewed as another tool to support the M&FM team in its goal of reducing algorithm risks and uncertainties to ensure mission success. The utility of the MATLAB® SFE is enhanced by its semantics, ease of interpretation, and processing execution of the modeled components. It is a deterministic method as the execution order among parallel states or outgoing transitions in a diagram is always sequential and fixed[5]. The notation schema defines a set of objects and rules that govern the relationships between the modeled objects providing a common language to communicate the design information, which is cross checked using other methods such as the carefully crafted UML-SysML M&FM algorithms, C^{++} prototype development and unit testing of those same algorithms within the M&FM team, FSW peer reviews, and the VMET results. Further system behavior can be captured by linking a state machine model to a physics-based model of the system dynamics [9] , which is being considered in forward work by the state flow NASA team.

State machines can be used as a computational model for designing and implementing software as is the case for the SLSP (shown in the next section), a technique commonly used for high-reliability software, especially flight software[9]. The executable state machines provide a framework that allows M&FM system engineers to analyze a system as opposed to simply describing it. This brings several advantages, including the ability to explore multitudes of "what-if" scenarios by specifying combinations of states of interest and observing the resulting progression of states [9]. Coupled with MBE methods, the semantics between both technologies provide a unique platform for notation and design interpretation and implementation schema. The joint activities between the NASA M&FM modeling team, C++ prototyping team (for VMET), and the state flow team have provided a rich platform to capture a variety of issues early in the design cycles of the M&FM algorithms. These methods have also prepared the M&FM team to be part of the FSW code peer review teams to further ensure software design and implementation of the M&FM algorithms.

II. The State Analysis Model (SAM)

The overall model comprising the SLS subsystems and the M&FM algorithms is termed the State Analysis Model or SAM as stated in the prior section and is depicted in Figure 3 below showing the components comprising the Plant and Controller state flow modules. Full details are sensitive but unclassified (SBU) and beyond the intent of this paper but are described further in an internal NASA document not publicly available[7]. Using all available data from various sources within NASA (e.g., concept of operations documents, schematics, vendor documentation, M&FM algorithms), each of the SLS subsystems comprise the Plant and the M&FM algorithms that comprise the Controller were modeled using the SFE constructs in MATLAB®. This was necessary in order to provide the crucial interactivity communication between the SLS subsystems and M&FM algorithms. Each of the SLS subsystems are modeled to a level of fidelity necessary to exercise each of the M&FM algorithms[7].

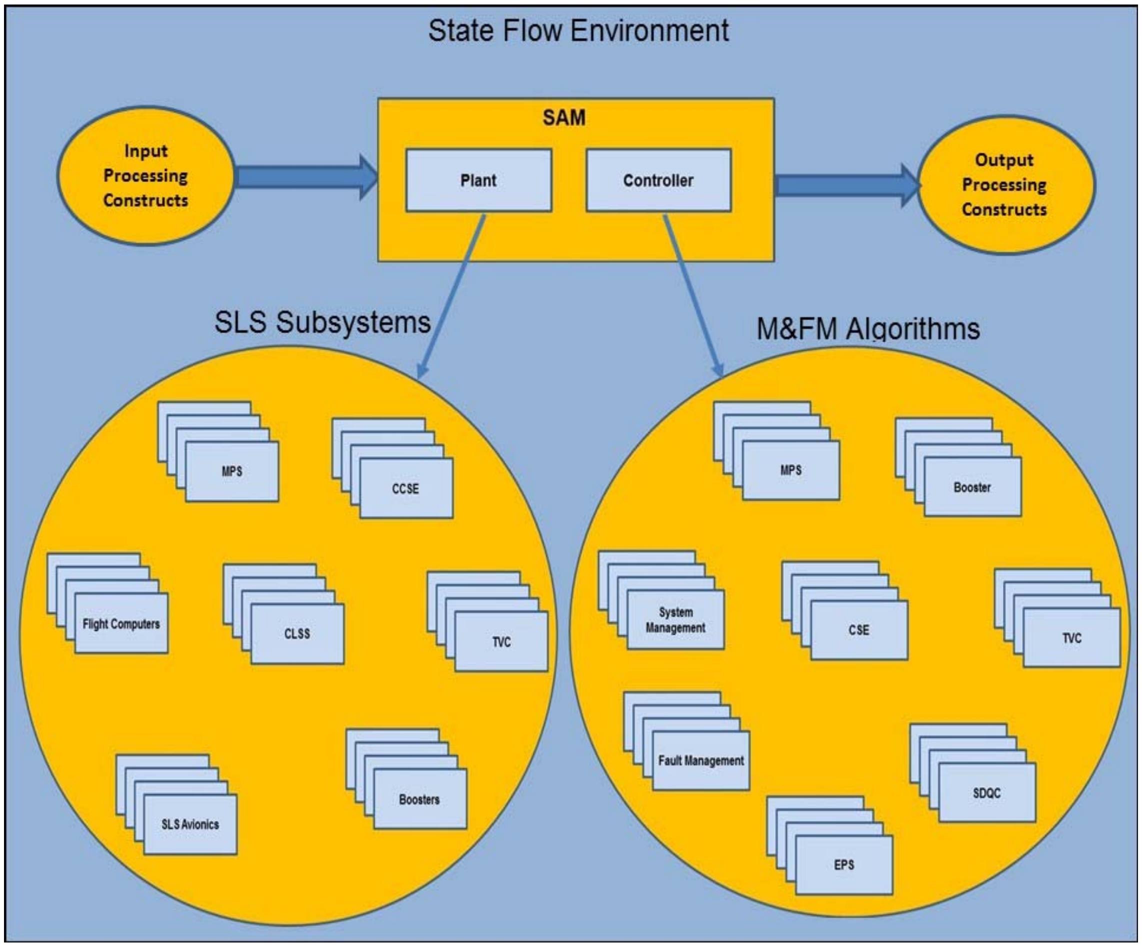


Figure 3. SAM Subsystem and M&FM Modules

In Figure 3 for the SLS subsystems depicted above is the State Flow Environment (SFE) which encompasses the SAM and the custom constructs for the input commanding, timeline reference, and output processing reporting, presented shortly. The Plant module, part of the SAM module, encompasses all the required avionics hardware state flow machines for interfacing with the M&FM algorithms which are represented within the Controller module of the SAM. This can also be viewed as the software component by which the M&FM algorithms are implemented. State machine components within the Plant module are shown on the left diagram. The Main Propulsion System (MPS) module hosts the individual state machines for key components such as valves and sensors. The CLSS module represents the control and sensor module state machines for the liquid oxygen (LO2) and liquid hydrogen (LH2) propellant tanks. The Booster module encompasses components such as the flight safety system, rock and tilt actuator (thrust vector control), control and power units, and other related state machines. The CCSE module hosts the valve driver module and power supply state machines for the MPS solenoids for the valves such as for the propellant prevalves, helium isolation valves, and the propellant tank pressurization valves. The Flight Computer module hosts the command and telemetry processing state machines and others. The SLS Avionics module hosts the remaining state machines such as flight buses, command and telemetry, and power and data acquisition state machines. The Controller module in the SAM hosts the state machines for the M&FM algorithms which were principally developed jointly by the M&FM and FSW teams via the integrated development teams (IDT) entailing the respective engineering disciplines[2].

For the input and output processing in Figure 3 above, execution and analysis of the SAM is performed by custom MATLAB® scripts, comparing the internal states at each time step with a set of disallowed state combinations after model execution. The processing and implementation details are out of scope for this paper but further detailed in the NASA internal document for the SLSP[7] and are briefly summarized hereas. For the input constructs in the SAM there is a nominal sequence generator where the input to the SAM is based on the (GSDO) SLS ground operations timelines in combination with the set of command templates developed by interpreting the system of concept of operations (conops) for each subsystem to generate a nominal command sequence. It is this conops that defines the steps required to perform a specific activity such as the process for opening or closing the propellant prevalves. The steps for each subsystem are then translated into a command template in the SAM which defines a set of commands for processing at a given time or condition. For testing off nominal conditions in the SAM, a set of fault scenarios can be generated through modification of the established nominal sequence, based on a set of critical scenarios for any given subsystem. Fault events are inserted for exercising select "what if" scenarios for failure propagation analysis. For this a set of fault conditions for specific times are generated and automatically executed in the SAM for each condition. This provides great flexibility for injecting faults for any subsystem(s) for select flight modes from pre-flight through post-flight operations. Part of the overall SAM execution process also entails the development of an input sequence for turning each command on the data bus into an index into an array at a specific time. This is generated from either the nominal or select fault scenario sequences via a MATLAB® script which then takes the array and places the command on the correct input bus at a specified time. Custom scripts were developed to conduct the execution and processing for output report generation along with visual interfaces and provided GUI for user selection of test cases. During model execution, MATLAB® stores information about state changes for all logged states. Logged signals are based on the states defined in the Rule Checker, a vital construct within the SFE. After model execution another script executes to analyze the stored state information for report generation for various views of system behavior and is also used with other MATLAB® scripts and tools within the SFE for further processing. There is also a Results Analysis script, a Timeline and State Report providing text based views of the logged states, all combined into a SAM Test Report. Example results are show in the next section.

For SLS each of the subsystems required many functions to perform the M&FM activities and analogous to software functions or subroutines, the SFE provides "subcharts" or children diagrams. For the UML modeling approach (in EA) functions are represented by drill down diagrams or composite diagrams exhibited by the eyeglass symbol as shown in the right portion of Figure 4 below, (also censored). Such reuse simplifies diagramming. This approach adheres to the NASA Orion (crew capsule) GN&C guide section 4.4.12 where it recommends MATLAB® Stateflow® for logic-intensive algorithms and MATLAB® script programming or Simulink for numerically intensive algorithms[8]. Graphical functions as used for SLS can be very useful for multilayered if-elseif-else constructs that might be difficult to follow as script text written using MATLAB® dot M files.

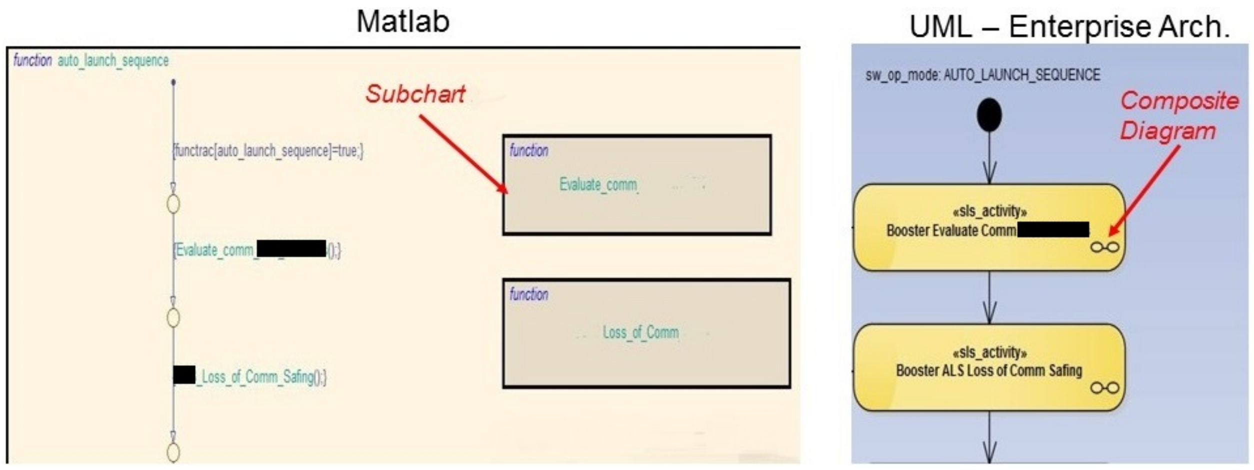
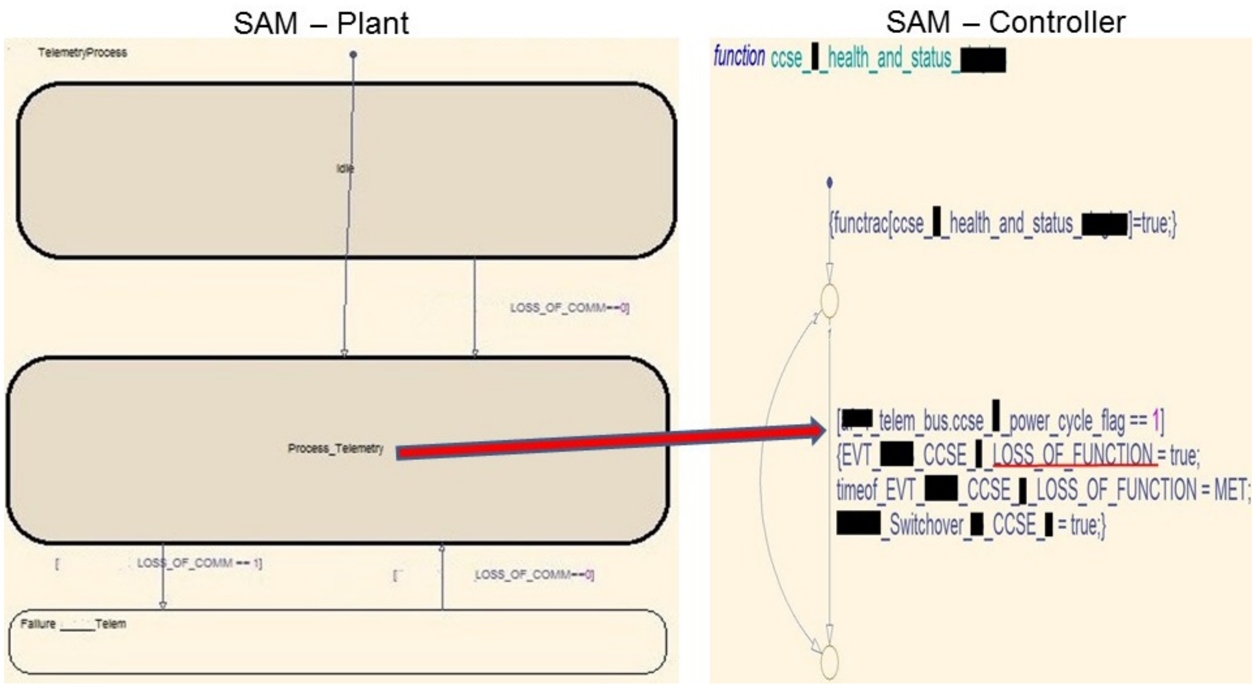

Matlab
function auto_launch_sequence
{functrac[auto_launch_sequence]=true;}
{Evaluate_comm[redacted]();}
[redacted]_Loss_of_Comm_Safing();}
Subchart
function
Evaluate_comm
function
Loss_of_Comm
UML – Enterprise Arch.
sw_op_mode: AUTO_LAUNCH_SEQUENCE
Composite Diagram
«sls_activity»
Booster Evaluate Comm[redacted]
«sls_activity»
Booster ALS Loss of Comm Safing

Figure 4 Function Representation in State Flow and UML Tool Enterprise Architect

So for a simple example of the interconnectivity flow in the SAM, Figure 5 below depicts a censored portion between the Plant and Controller modules for the CCSE power cycle flag status. Once telemetry detects the power cycle flag as signaled by the CCSE hardware within the "Process Telemetry" subchart of the Plant (left diagram in Figure 5 below), the guard condition in the Controller (right diagram in Figure 5 below) sets the loss of function event (also within its own subchart but at a higher level not shown).



```
function ccse_[redacted]_health_and_status_[redacted]

{functrac[ccse_[redacted]_health_and_status_[redacted]]=true;}

[[redacted]_telem_bus.ccse_[redacted]_power_cycle_flag == 1]
{EVT_[redacted]_CCSE_[redacted]_LOSS_OF_FUNCTION = true;
timeof_EVT_[redacted]_CCSE_[redacted]_LOSS_OF_FUNCTION = MET;
[redacted]_Switchover_[redacted]_CCSE_[redacted] = true;}
```

Figure 5. Sample Interactivity Flow between Plant and Controller Flow within the SAM

For the SLSP, it turned out that the number of states is less than the number of transitions between states which implies there are states with more than one transition entering into it or leaving from it. Furthermore, the quantity of data signals such as commands being much less than the number of transitions implies there are many transitions based on the state of other components, which is expected for a complex system like SLS[7]. For SLS the SAM models were configured to run using a fixed step, discrete solver with a time step matching the FSW frame schedule, via ode 3 Bogacki-Shampine[10]. Boolean pulse signals that oscillate between zero and one every processing frame were used as inputs in the SAM. Within SAM, events are treated as instantaneous occurrences, only transpiring when the signal's value changes from non-positive to positive or from positive to non-positive. This concept is similar to MATLAB® Simulink triggers, which cause subsystems to activate only when the trigger's value changes. In essence, specifying an event to be an input to the SAM causes the diagram to behave as a triggered subsystem. Transition conditions and input events are permitted by both Mealy[11] and Moore[12] machine standards[8]. The choice between the two styles is driven more by desired system behavior and if the algorithm is intended on executing at irregular intervals, i.e., condition based, then events are preferred. When adopting state flow for logic-intensive algorithm design, designers and developers are faced with a great deal of flexibility and choice in how M&FM algorithms are implemented. In MATLAB® there are four basic choices which are state transition definition, action definition, function representation, and action language selection[8]. For the SLSP a hybrid of event-driven and condition driven state machines exists out of necessity due to the complexity and diversity of the M&FM algorithms for each of the subsystems. Event-driven state machines are well suited for aperiodic and message-based applications whereas transition conditions work well for fixed-rate systems which is the case for some of the SLS deterministic subsystems (e.g., activities during the autonomous launch sequence from T-31 seconds through T-0 such as checking the full closure of the overboard bleed valves at a specific time). For SLSP a hybrid of state, condition, and transition actions have a set order of execution that should be understood in MATLAB® Stateflow® when combining them together, relying heavily on junctions, other states, commands, and conditions such as parametric thresholds being reached (e.g., pressure measurements), critical for M&FM methods. Thus, to be clear, for the SAM the choice for the machine standard for processing state flow diagrams was the default choice of the hybrid state machine that combines the semantics of the Mealy and Moore charts which is labeled as "classic" in MATLAB®.

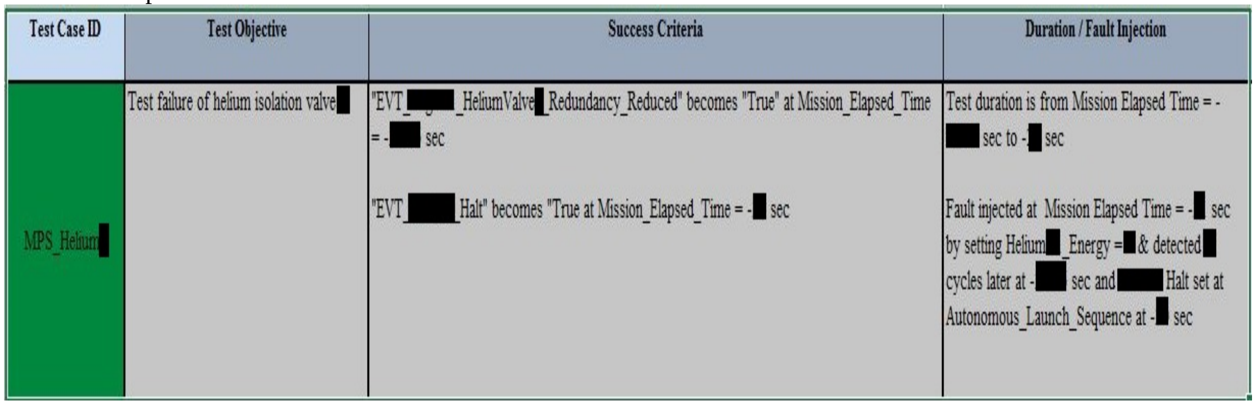
III. SAM Testing: M&FM Algorithms with VMET Test Cases

State Flow Environment activities have provided extensive analysis for testing the various products for the SLSP. Initial testing and analysis began with investigating the M&FM algorithms for several of the subsystems such as MPS, electrical power system (EPS), System Management, core stage engine (CSE), and TVC. The SAM team then proceeded to investigate the VMET test cases for the MPS. This section will also address the test cases against the Monitored Conditions and Response Table (MCaRT) and the SIL test cases. What is clear from this work is that testing and validating critical fault management designs before implementing them in a real platform has proven the utility of the MATLAB® Stateflow® product as a viable tool for quick look analysis of system design performance for fault management systems. This supports claims from the MBE community that using MBE methods for performing validation and verification during the early stages of development provides a common platform for communications between subsystems, system engineers, and stakeholders[3,13]. The SAM has better prepared the M&FM team for its next phase of design and development activities such as M&FM algorithm updates, for Exploration Upper Stage (EUS) for SLS, or support of other future space applications. It has also prepared the M&FM team for its activities in serving as FSW peer reviewers of M&FM algorithms implemented in CMMI level III certified FSW. All testing and results has been captured in the TRAC problem reporting system which has further enabled the M&FM and state flow system teams to keep track of issues and resolve them as the M&FM design evolves and issues are resolved within the IDTs. The initial intent of VMET is risk buy down further instilling system reliability and assurance of the M&FM algorithms in a platform exterior to the other NASA higher fidelity testing organizations such as the software development facility (SDF) and the SIL which are still progressing [2].

For testing the SAM in the SFE, the MATLAB® tool resides on a Dell 64 bit Windows 7 laptop where the SAM takes approximately 120 seconds to execute a run of a candidate mission launch profile[7]. This is accomplished utilizing the SLS timeline of events and command sequences for on pad operations up to thirty-six hours prior to launch, through ascent, and throughout the end of mission operations. Within the SFE the SAM is capable of injecting failures at select mission critical points per the mission timeline during all on pad and flight modes. The SAM has also exhibited the ability to manifest single and multiple failure behavior without ground interaction from the GSDO which is planned forward work for modeling in the SFE[7]. As presented in section II, failures are injected within the nominal command sequence. SAM analysis requires that the nominal command sequence, with no faults injected, includes no off-nominal events or Rule Checker violations and serves as a baseline from which to springboard to many types of off nominal scenarios[7]. From this, as summarized in section II, the failures are injected within the nominal command sequence, results summaries presented later in this section.

Using SAM to analyze the VMET test cases provided a rapid assessment of the test procedures prior to their execution in VMET enabling the VMET prototype developers and VMET test conductors to debug their VMET test procedures and algorithms, further adding value by increased understanding and thus reducing errors and forward testing cost in the more resource-intensive VMET environment which is still evolving to integrate the physics based models. SAM VMET analysis, to date has consisted of approximately 50 VMET test cases generated for MPS, which was the initial focus due to the quantity and complexity and criticality of those algorithms. The SAM analysis closely approximated the test and corresponding test objectives for each VMET MPS test case. The outcome of each SAM test produced a pass/fail result similar to the VMET test procedure[7]. A sample MPS test case for failing a select helium isolation valve to generate the corresponding M&FM event is shown below in Table 1 emphasizing the need for a clear Test Objective, Success Criteria, and fault injection method during a select time or time frame from the mission timeline.

Table 1. Sample Test Case for MPS for VMET



Test Case ID	Test Objective	Success Criteria	Duration / Fault Injection
MPS_Helium[redacted]	Test failure of helium isolation valve[redacted]	"EVT_[redacted]_HeliumValve[redacted]_Redundancy_Reduced" becomes "True" at Mission_Elapsed_Time = -[redacted] sec "EVT_[redacted]_Halt" becomes "True at Mission_Elapsed_Time = -[redacted] sec	Test duration is from Mission Elapsed Time = -[redacted] sec to -[redacted] sec Fault injected at Mission Elapsed Time = -[redacted] sec by setting Helium[redacted]_Energy =[redacted] & detected[redacted] cycles later at -[redacted] sec and [redacted] Halt set at Autonomous_Launch_Sequence at -[redacted] sec

In addition to executing the SAM on the M&FM algorithms, the VMET test cases provided a guideline procedure on testing the MPS subsystem in an integrated environment providing a forecast of expected performance. In general the process for testing and reporting was the same regardless of the subsystem or the test case. The process began with execution of the SAM and the subsequent analysis of the results. The output of this analysis was then used to generate a TRAC trouble ticket for each discovered issue. As tickets populated the TRAC system it became obvious that categories and severity ratings were needed enabling the development of attributes such as defect type, milestone, and priority. Once tickets were generated by the state flow team, the M&FM team point-of-contact (POC) reviews each ticket for acceptance. Rejected tickets were closed and accepted tickets were vetted with the respective subject matter experts which usually entailed discussions with the IDT lead and FSW developer. For tickets that depended on resolution of design but required more time due the design cycles of the SLSP, tickets were re-categorized as “pending” by the M&FM POC or the SAM lead. Once the design is updated through a formal SLSP process, the tickets are either closed or remain open if the issue is not fully resolved. In this case the ticket remains open until forward work is completed. This happened several times throughout the previous design lifecycle. Table 2 below summarizes the TRAC tickets for each of the subsystems with VMET as its own separate entry:

Table 2. TRAC Results of VMET and Subsystem SAM Testing [7]

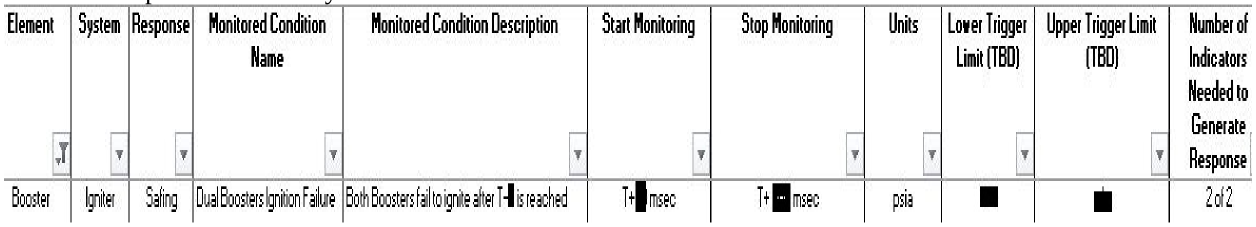
System	Test Cases Passed	Test Cases Failed – Tickets Generated	Tickets Closed / Open
VMET (MPS)*	41	7	5/2
MPS	**	48	44/4
Booster	**	20	15/5
EPS	**	7	4/3
Core Stage TVC	**	11	6/5
Avionics	**	2	1/1
Core Stage Engine	**	23	22/1
Fault Management	**	3	0/3
Mission Management	**	6	6/0
Time Management	**	2	2/0

* 2 cases were not tested due to lack of model fidelity in the SAM, forward work
** Test cases stemmed from nominal testing and algorithm specific off nominal testing, not VMET test cases

A. MCaRT and System Integration Lab (SIL) SAM Testing

The purpose of this effort was to use the SAM to verify the listed conditions in the MCaRT are in agreement with the intent of the M&FM algorithms, identify if any of those conditions are detected outside the specified monitoring times, and if any false negatives existed (i.e., monitored condition was not detected when detection was expected)[7]. The MCaRT lists the various conditions that are monitored onboard the vehicle, along with associated design information. A sample of a MCaRT entry is shown below in Table 3 (also censored).

Table 3. Sample MCaRT Entry



Element	System	Response	Monitored Condition Name	Monitored Condition Description	Start Monitoring	Stop Monitoring	Units	Lower Trigger Limit (TBD)	Upper Trigger Limit (TBD)	Number of Indicators Needed to Generate Response
Booster	Igniter	Safing	Dual Boosters Ignition Failure	Both Boosters fail to ignite after T-[redacted] is reached	T+[redacted] msec	T+[redacted] msec	psia	[redacted]	[redacted]	2 of 2

The SAM analysis of the MCaRT conditions resulted in some disparities between the monitoring times specified in the MCaRT and the actual performance of the M&FM algorithms, as specified in the M&FM Model. To date about 19% of the MCaRT entries have been tested resulting in about 85.5% of the test cases passing, generating about 8 TRAC trouble tickets which were eventually closed once the updated MCaRT was released in forward work[7]. Although only a small subset of the MCaRT had been tested, the findings so far have illustrated the value of the SAM for crossing checking other M&FM products against the M&FM models.

The SAM was also used to support the test cases developed for use in the SIL. The SIL is a high-fidelity testbed, integrating the actual FSW with a mix of real and simluated SLS hardware and environments. Using the analogy of testing the VMET test cases, a like approach was taken and also gave performance foresight of the M&FM algorithm test cases in a hardware-in-the-loop setting. There were approximately 93 test cases of which 45% were tested in the SAM resulting in 27% of those test cases passing and generating four TRAC trouble tickets for the M&FM team to investigate since select M&FM team members are actively involved in defining SIL test case development processes, including the state flow lead. To date those same four TRAC tickets are still in the process of being resolved due to ongoing SIL test case definitions.

To summarize the results of executing the SFE on the M&FM, MCaRT, and SIL products, as reported[7], failures injected during on pad activities (prior to ALS), provides a fuller understanding of system behavior during ALS with loss of certain sensor data or control function. This knowledge of system behavior under specific failures will be supportive to the M&FM team in supporting GSDO procedure development. Likewise, failures injected during ALS with observed instantaneous changes in behavior also provides insight and can potentially be used to improve M&FM system design in forward work such as for EUS for future SLS missions. Failures were also injected in the SAM during core stage booster flight, and it was observed that the vehicle simulation exhibited sufficient resilience to continue controller flight in a state of lost or degraded functionality, identifying potential areas for improving M&FM design fidelity in the SAM Plant and Controller modules. The main areas of focus for failure injection were in loss of communications, loss of power, and loss of CCSE functionality across all vehicle modes. Loss of communication resulted in stale data. Loss of power helped determine vehicle behavior when power functionality is lost for a component, e.g., due to a battery unit failure. And loss of CCSE functionality illustrated expected redundancy management functions for the MPS subsystem keeping critical valves functional[7].

IV. Forward Directions and Summaries

For the SLSP, plans include expanding the use of the SFE-SAM capability to search for interactive failures, identifying risks in pre-launch procedures, assessing design changes (including hardware, software, command sequencing, and parameters), and performing post-flight analysis in a common framework[7]. Current plans include extension to improving operational procedures, such as Operations and Maintenance Requirements (OMRs) being developed by NASA (GSDO) to ensure procedures do not violate rules placing the SLS vehicle into any hazardous conditions. For post flight applications, plans are to investigate utility to track observable state transitions and identify potential hidden unobserved hazardous states for fleet supportability[7]. Furthermore, current activities are underway to expand the Rule Checker capabilities to include rules from ongoing hazard analysis and launch commit criteria (LCC) development activities taking place across NASA. The SAM is also currently being extended to the EUS program as stated above. Beyond SLS, the SFE-SAM capability is considered to be applicable, adaptable and expected to be highly beneficial in designing robust systems for use on future space ventures such as crew habitat's specialized payloads, proximity operations, rovers, crewed and robotic deep space missions, and large scale entry, decent, and landing (EDL) to name a few.

In summary, the complexity of space missions has dramatically increased with more of the critical mission and fault management aspects of a spacecraft's design being implemented in software. With sound development practices, the end product of reliable FSW contributes to enhanced assurance for mission success via robust fault management of physical system components, its computing platform, mission software, and to crew safety via validated caution and warning, safing, and abort algorithms[1,2]. With the added functionality and performance required by the FSW to meet overall vehicle and M&FM system requirements, the robustness of the software algorithms must be fully validated. While fault management software is needed to manage fault tolerance, safing, caution and warning, and aborts, that added software is also a new source of failure. To date, the only remedy to reduce such software and integrated system uncertainties is testing from several independent diverse perspectives: testing has been the most widely applied software quality assurance technique over many decades. Due to its successful practical applications, considerable software engineering research efforts has occurred for improving the effectiveness of testing in order to scale the techniques for handling ever more complex software systems for spacecraft[14]. However, traditional validation methods of simulation and testing are continually being challenged to adequately cover the needs and concerns in this evolving environment within cost and schedule constraints. As a result, model-based engineering methods are emerging as a powerful validation technique for mission critical software[6]. This application of state flow methods, via the new SFE-SAM for the SLSP, has proven to be very beneficial for quick look analysis (i.e., crystal ball effect), into early integrated system behavior and assessment of the M&FM algorithms with the modeled LV subsystems for the SLSP. Since simulation does not provide exhaustive verification of system requirements, having a variety of simulation tools provides a variety of perspectives on performance provides additional confidence in the M&FM design, as supported by claims from industry and academia that model-based engineering is key to improving system engineering and embedded software engineering for spacecraft systems[1].

Appendix

State: A state describes a mode of an event-driven system. The activity or inactivity of the states dynamically changes based on events and conditions. Every state has a parent. In a Stateflow® diagram consisting of a single state, that state's parent is the Stateflow® diagram itself (also called the Stateflow® diagram root). States have labels that can specify actions executed in a sequence based upon action type. The action types are entry, during, exit, and on. Stateflow® provides two types of states: parallel (AND) and exclusive (OR) states.

Parallelism: Parallelism represented with AND (parallel) states.

Exclusive (OR) states: These are used to describe modes that are mutually exclusive cases, links one object to another. One end of a transition is attached to a source object and the other end to a destination object.

Source: The source is where the transition begins and the destination is where the transition ends.

Transition: A transition label describes the circumstances under which the system moves from one state to another. It is always the occurrence of some event that causes a transition to take place.

Events: Events drive the Stateflow® diagram execution but are non-graphical objects and are thus not represented directly in a Stateflow® chart. All events that affect the Stateflow® diagram must be defined. The occurrence of an event causes the status of the states in the Stateflow® diagram to be evaluated. The broadcast of an event can trigger a transition to occur or can trigger an action to be executed. Events are broadcast in a top-down manner starting from the event's parent in the hierarchy. In Stateflow® semantics there is at most one event active at a time [1]. In Stateflow® it is not possible to select one enabled transition non-deterministically.

Hierarchy: This enables the organization of complex systems by defining a parent and offspring object structure. A hierarchical design usually reduces the number of transitions and produces neat, manageable diagrams. Stateflow® supports a hierarchical organization of both charts and states.

Acknowledgments

This work was funded under the JACOBS ESSSA Group for NASA under contract NNM12AA41C. The principal author thanks the fellow co-authors for their contributions of their combined knowledge of LV health management principles, M&FM processes paper, and state flow principles. The authors especially thank the leadership of Mr. Jonathan Patterson[†††] and Mr. David Teare[‡‡‡]. A special acknowledgement and thanks is granted to 2016 NASA Intern Ms. Kati Williams from the University of Missouri - Kansas City for her modeling of the M&FM algorithms for the RGA and RINU subsystems and providing feedback and graphics support for this paper. Further acknowledgement is also granted to the Flight Software Group and participating organizations[§§§] in all M&FM activities. In conclusion the authors especially acknowledge the overall M&FM team[****] for their friendly team spirit and cohesive efforts for addressing the M&FM needs for each of the SLS launch vehicle subsystems for the SLSP.

[†††] SLSP NASA lead for the M&FM Team in ISHM, Spacecraft and Vehicle Systems Department, EV43

[‡‡‡] Implementation and Test Team lead in ISHM, EV43

[§§§] Such as Crew Office, Ground ops, SPIO, Avionics, GNC, KSC, JSC, GRC, ARC, Booster, Liquid Engines, and S&MA

[****] VMET, analysis, algorithm developers, schedule leads, IDT leads, prototype developers, and all administrative and project support personnel

References

[1] P. Feiler, J., J. Goodenough, A. Gurfinkel, C. Weinstock, L. Wrage, “Reliability Validation and Improvement Framework,” Special Report CMU/SEI-2012-SR-013, Software Engineering Institute, Nov. 2012

[2] L.Trevino, S. Johnson, J. Patterson, D. Teare, “A Vehicle Management End-to-End Testing and Analysis Platform for Validation of Mission and Fault Management Algorithms to Reduce Risk for NASA’s Space Launch System,” 2015 International Test and Evaluation Association Test Technology Review, Huntsville AL, November 2015

[3] M. Briggs, N. Benz, D. Forman, “Simulation-Centric Model-Based Development for Spacecraft and Small Launch Vehicle,” 32st Space Symposium, April 2016

[4] National Highway Traffic Safety Administration Toyota Unintended Acceleration Investigation, Technical Support to the National Highway Traffic Safety Administration on the Reported Toyota Motor Corporation Unintended Acceleration Investigation, January 2011

[5] C. Chen, J. Sun, Y. Liu, J. Dong, M. Zheng, “Formal modeling and validation of Stateflow® diagrams,” Online publication, Springer-Verlag, June 2012

[6] P. Pingree, E. Mikk, G. Holzmann, M. Smith, D. Dams, “Validation of Mission Critical Software Design and Implementation Using Model Checking,” The 21st Digital Avionics Systems Conference, October 2002

[7] M&FM Design Analysis & Performance Assessment Document Volume 4: State Analysis, SLS-RPT-087-04, NASA internal document for the SLS Program, June 2016

[8] W. Campbell, M. Anthony, B. Petteys, “Understanding Model and Code Behavior for State Flow Constructs,” The Mathworks Inc. 36th Annual AAS Guidance and Control Conference, American Astronautical Society, Breckenridge CO. February 2013

[9] J. Harris & A Patterson-Hine, “State Machine Modeling of the Space Launch System Solid Rocket Boosters,” NASA Aeronautics Scholarship Program – Internship Final Report, Aug. 2013

[10] https://en.wikipedia.org/wiki/Bogacki%E2%80%93Shampine_method

[11] G. Mealy, A Method for Synthesizing Sequential Circuits. Bell System Technical Journal, Vol. 34. Bell System, 1955.

[12] E. Moore, Gedanken-experiments on Sequential Machines. Princeton, NJ. Princeton University Press, 1956.

[13] A. Homar, AOCS Fault Detection, Isolation, and Recovery, Master’s Thesis, Lulea University of Technology, Department of Computer Science, Electrical and Space Engineering, Friedrichshafen, Germany, Sept. 2014

[14] H. Yenigin, C. Yilmaz, A. Ulrich, “Advances in Test Generation for Testing Software and Systems: An Introduction to selected papers from ICTSS 2013,” International Journal on Software Tools for Technology Transfer, pp. 1-5, October 2015