

# State-Based Behavior Modeling of the Integrated SLS-MPCV System

August 14<sup>th</sup>, 2012

Kevin H. Bonanne<sup>1</sup>

*with Mentor*  
Dr. Oleg V. Sindiy<sup>2</sup>

*and Co-Mentor*  
Dr. Otfried Liepack<sup>3</sup>

*Jet Propulsion Laboratory, California Institute of Technology  
4800 Oak Grove Drive, Pasadena, CA 91109*

**In NASA's effort to foster a human spaceflight capability beyond Earth's orbit, two space systems are being developed – the Space Launch System (SLS) and the Multi-Purpose Crew Vehicle (MPCV). As of this time, the interactions between the two systems during launch are not fully detailed. To remedy this situation, a Systems Engineering approach utilizing models was developed to investigate the behavior of the integrated SLS-MPCV stack during ascent and abort situations. Specifically, this innovative approach combines aspects of Model-Based Systems Engineering (MBSE) and state analysis to simultaneously model the physical, functional, and behavioral aspects of systems. This approach focuses solely on the interactions between the systems, leaving much of the internal workings of either system at a logical level (i.e., black box). By utilizing this newly defined approach, a behavior model for the integrated SLS-MPCV stack was developed, emphasizing only the subset of interactions between the systems that impact behavior. Finally, analysis is performed within the model to investigate requirements gaps and examine the execution times of key behaviors related to various ascent phases and abort scenarios. The work described in this paper is merely a portion of the outlined effort being undertaken for this project; only a segment of the SLS-MPCV system behavior will be described.**

## I. Introduction

**I**N the last 40 years, no human has explored farther than low Earth orbit (the farthest being approximately 556 kilometers above sea level)<sup>i,ii</sup>. To continue the spirit of exploration that NASA has always exhibited, the Multi-Purpose Crewed Vehicle (MPCV) and Space Launch System (SLS) are being developed as the future deep-space capsule and heavy-lift rocket, respectively, to take humans out of Earth's gravity well and on to exciting new places in our solar system<sup>iii,iv</sup>. These systems are currently in development at NASA centers as well as contracted facilities around the country.

In order to better understand how these systems will behave when together, JPL has been tasked with leading a multi-center effort to generate a behavior model for the SLS-MPCV integrated stack. As described in the charter for this project, the goal of this task is to create a “model of the behavior of the system at the interfaces [to] provide critical detail of the system's behavior in nominal and off-nominal scenarios.”<sup>v</sup> Further study goals include establishing an approach for performing cross-system behavior modeling and identifying gaps within the working

---

<sup>1</sup> JPL Affiliate, Section 318A, Jet Propulsion Laboratory, funded by Student Internship Program.

<sup>2</sup> JPL Employee, Section 318A, Jet Propulsion Laboratory.

<sup>3</sup> JPL Employee, Section 318E, Jet Propulsion Laboratory.

Interface Requirement Documents (IRDs), which detail the interactions between SLS and MPCV, but may have inconsistencies because of the separated nature of the two programs.

The behavior modeling process developed to accomplish this task has five distinct parts. First, information about the system is gathered in the form of system requirements and capabilities, derived from Concept of Operations documents. If there is a specific mission or scenario that the system is performing, the phases of that mission should also be defined. With the details of the system and mission known, the next step is to detail the physical aspects of the system. The properties of the physical system are established by performing a state analysis and mapping how these states affect each other. From a state effects diagram, a functional flow is created to show the transitions between the various states, in effect detailing the behavior of the systems. These functions can also map to the physical components that perform them. With these three major aspects defined, analysis can be performed to examine various aspects of the system (including gaps in requirements). All modeling is performed using the Systems Modeling Language<sup>vi</sup> notation within the MagicDraw tool<sup>vii</sup>.

The remainder of this paper will provide further detail on the behavior modeling process that was developed. While discussing the methodology, some basic abstract examples will be provided to briefly explain how the process works and what many aspects of it will look like in SysML. This can be found in Section II. Section III describes the application of this process to the SLS-MPCV integrated stack, specifically examining behaviors during launch and abort scenarios. Lastly, results and future work will be provided to conclude the paper in Section IV.

## II. Behavior Modeling Methodology

The methodology of this process was established by utilizing ideas from Model-Based Systems Engineering<sup>viii</sup> – specifically the Systems Modeling Language (SysML)<sup>vi</sup> notation for system representation – and state analysis<sup>ix</sup>, which has roots in control theory. MBSE provides much of the framework for defining a system in terms of its physical and functional components, but lacks formal definition in the area of behavior modeling. On the other hand, state analysis primarily captures behavior, but lacks a formal modeling process. The unique methodology selected for this project seeks to bring together elements of both fields to fully capture the behavior of a system in a model. Using this combined approach, analysis can then be performed on the system. Generalize examples of each step in this process will be provided in this section; however, an analysis example will only be shown at the end of Section III, due to the necessity of distinct details for analysis.

### A. Requirements, Capabilities, and Mission Phases

To begin, knowledge about the system(s) must be acquired. This approach recognizes two forms of data from which to generate the model. First, the capabilities outline the intended features and functionality for a system. Capabilities outline what the system should be able to do and thus are a solid starting point from which to begin creating the model. A capability may be worded as, “System A will send telemetry to System B”. This capability provides the facts that there are two systems, A and B, within a domain; both systems can hold the same telemetry; system A sends telemetry to system B; and system B receives this telemetry. Capabilities may be drawn from several documents or models detailing how a system operates (the later example draws capabilities from a single source).

The second primary source of data comes in the form of requirements. Requirements explain what a system must be able to do and often contain constraint information. A requirement should always have the form, “System A shall...”. As requirements are often used within systems engineering to ensure a system is designed correctly (within all constraints), it is the duty of the model to verify that the provided set of requirements is complete. Thus, requirements are used both as a source of information for the model and are analyzed by the model for completeness.

At this point, it is also important to outline the phases of the mission that is being modeled; e.g., testing and integration phases vs. launch phases vs. on-orbit operations and so forth. Mission phases provide a storyboard of what states the system(s) can transfer between. These changes are defined by a difference in physical or functional aspects of the systems (i.e., the system looks or acts differently). Because of these changes, certain requirements and capabilities will only be applicable for certain phases. Mission phase information can be captured in the form of a state machine diagram (*stm*) in SysML (see Figure 1). Having gained knowledge on the system and captured details involving the mission, the bulk of the modeling of the system can be performed.

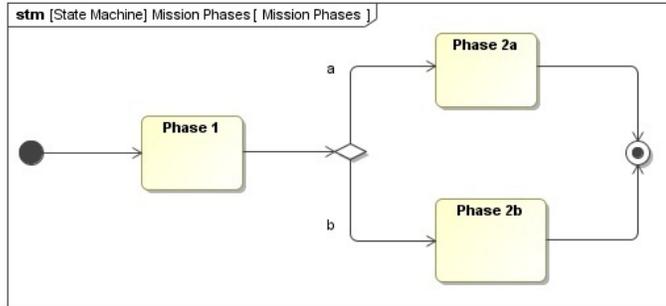


Figure 1: Mission Phases

### B. Physical Description

The first viewpoint of the system to capture is the physical decomposition. This entails identifying the high-level system(s) and its lower-level components. Furthermore, the physical connections between these systems must be defined (e.g., adapters, communication lines). Often a physical decomposition is created in SysML using a block definition diagram (*bdd*) to show the decomposition and an internal block diagram (*ibd*) to show any connections. It is most important to only decompose a system to the highest level required to answer the study goals of the model. This scoping issue will occur for each of the major aspects of this modeling process (functional and state definition). An example physical decomposition is shown in Figure 2.

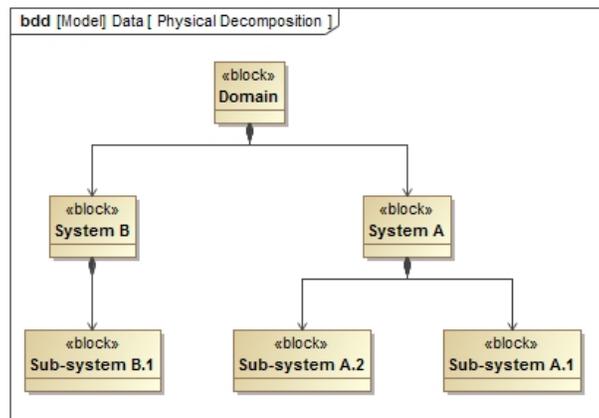


Figure 2: Physical Decomposition

In this example, the two systems shown are entirely independent, but operate under the same domain. By utilizing this arrangement, an *ibd* can be created within Domain to show any physical connections between System A and System B. In Figure 3, there are two modes of communication between Systems A and B, one wired and one wireless. Note that these connections may not be active over every phase of the mission. However, the physical decomposition view shows all systems and their possible connections even if they are only available at certain times. In this way, it functions as a master view of all physical systems.

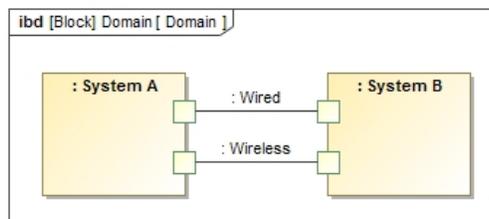
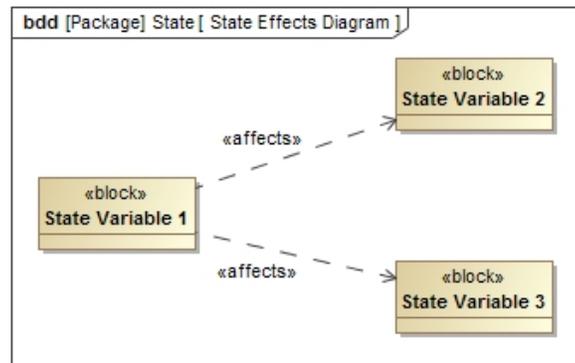


Figure 3: Physical Connections

### C. State Analysis

Following the physical description of the system, the next viewpoint to define is the properties of those systems via a state analysis. Based on what properties of the system are important to capture, various states are outlined and related in a state effects diagram (SED). A state effects diagram draws from one state variable and maps all other state variables, measurements, or commands that affect the original state. An example of a basic state effects diagram is presented in Figure 4.



**Figure 4: State Effects Diagram**

A state variable is a property of a physical element and may be operated on by a function. A state measurement is an approximation or estimation of a state variable – estimation being an amalgamation of approximations (e.g., position estimate being composed of altitude, accelerometer, and pressure measurements). A state command designates the intent to change a state variable. For example, a command to “Open Valve A” will change the “Valve Position” state variable from closed to open. A state measurement would approximate the valve to be open, closed, or unsure if there is not enough information to determine the valve state.

This process may have to be revisited multiple times in order to ensure that all necessary states are captured. Furthermore, it can be useful to perform the state analysis and functional decomposition simultaneously, establishing states while outlining the functions that operate on them to ensure that the scoping on both viewpoints is consistent.

#### **D. Functional Definition**

The functional definition provides the final major viewpoint in describing the system. A functional definition consists of a functional decomposition and functional execution, which define how the functions are organized and how they are performed, respectively. A function is defined as an operation to transfer between an inputted set of state variables, messages, or commands to an outputted set.

To begin a functional decomposition a high level function must be defined. Sometimes such high-level goals are defined as goals or operations in other contexts<sup>x</sup>, but for simplicity, they will fall under the mantle of functions in this paper. This top level goal is fragmented into the sub-functions that may be performed in order to accomplish their parent function. This process continues until a sufficient level of abstraction is met. In SysML, a functional decomposition is often portrayed within a *bdd*.

A functional decomposition follows similar rules as a physical decomposition, aiming to decompose only to the highest level required to fulfill the study goals of the model. Similarly, each function should be able to map to a system. An example of a functional decomposition can be seen in Figure 5.

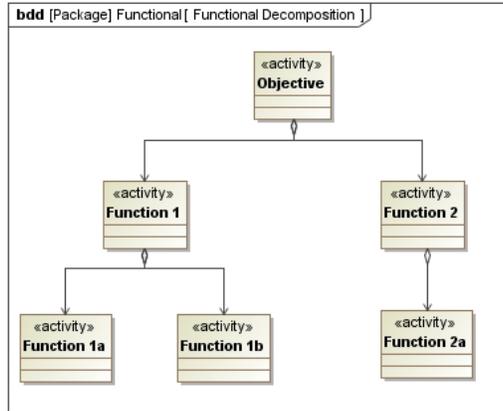


Figure 5: Functional Decomposition

To completely capture the functional viewpoint of a system, a functional execution must be performed. The functional execution describes the process that occurs within each function (filling the black box) and is performed for each function. Parent functions will contain their children and the logic that transitions from one function to another. This viewpoint is similar to a classic Functional Flow Block Diagram (FFBD) but also contains the object flow (state variables, measurements, or commands) between each function and can contain constraint information as well. Functional executions are shown as activity diagrams in SysML, as seen in Figure 6.

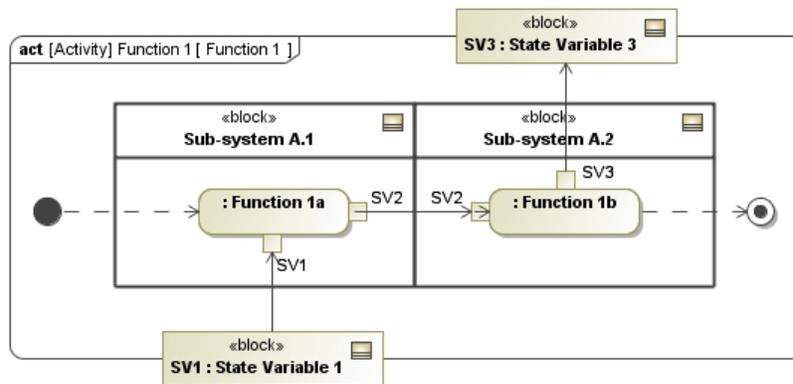


Figure 6: Functional Execution

### E. Combined Viewpoints and Verification

Between the major viewpoints that have been described, it is important to connect the various elements when possible. This creates a cohesive model and allows for analysis to be performed between the viewpoints. Numerous actions can be performed to create a cohesive model. First, functions and physical elements can be connected by containing the functions within the physical blocks in the containment tree (see MagicDraw tutorials<sup>xi</sup> on how to do this); this will automatically create a relationship between the two elements. On the other hand, allocating functions to swim lanes owned by their performing element will provide a more graphical relationship.

As state elements are already used in the functional viewpoints, it is important to verify that all interactions that occur within those functional diagrams are also represented by an “affects” relationship between the states (on the State Effects Diagram). Likewise, if information is transitioned between two swim lanes in a functional viewpoint, there must exist a physical connection through which information can flow between those systems.

Finally, a combined viewpoint showing aspects of the physical, functional, and state viewpoints can be created using an *ibd*. In order to accomplish this, the *ibd* should show the systems and their connections. From this standard starting point, functional execution diagram(s) (containing state information) can be placed upon each system block showing what function can be performed within each block. This type of diagram makes it easier to examine a part of the behavior of a system(s) and how it maps to the physical aspect of that system(s), namely the interactions between/within a system(s). For the application problem presented later, this viewpoint was ideal for allocating and examining requirements.

### III. SLS-MPCV Application Problem

As stated in the introduction to this paper, the modeling methodology above was used to capture the behaviors of an integrated SLS-MPCV system, specifically in launch and abort scenarios. This section will proceed through the steps taken to model this system and walk through a limited example using the actual system information. As previously stated, the process will require capturing data on the system and mission, defining the physical viewpoints, mapping out the state variables that are of importance to the problem, capturing behavior through the functional viewpoints, and performing analysis on the model.

#### A. Requirements, Capabilities, and Mission Phases

There are two major sources where the knowledge about the SLS-MPCV integrated stack existed. First, an Integrated Mission Analysis (IMA) report<sup>xiii</sup> by the Exploration Systems Development division<sup>xiii</sup> defines a number of capabilities that the SLS-MPCV integrated stack will have. This report draws from the Concept of Operations for the SLS, MPCV, and ground systems, each of which details the systems and their required interfaces. The other source of knowledge about the system comes in the form of the Interface Requirement Documents (IRDs), which provide more details about each of the expected interfaces. Though part of the purpose of this model is to verify that the IRDs are complete and accurate, they are still a source of knowledge that should be captured in the model. All capabilities and requirements are imported into tables in the model and then related to specific model elements to allow for gap analysis (see Section III-E).

It is also important to outline the phases of the mission that is being modeled. For launch scenarios, booster separations and spacecraft deployments all indicate changes in phase. Mission phase is also a way to distinguish between nominal, off-nominal, and abort scenarios – capturing each as a different mission phase or sub-phase. The mission phases identified for this application are shown in the Figure 7.

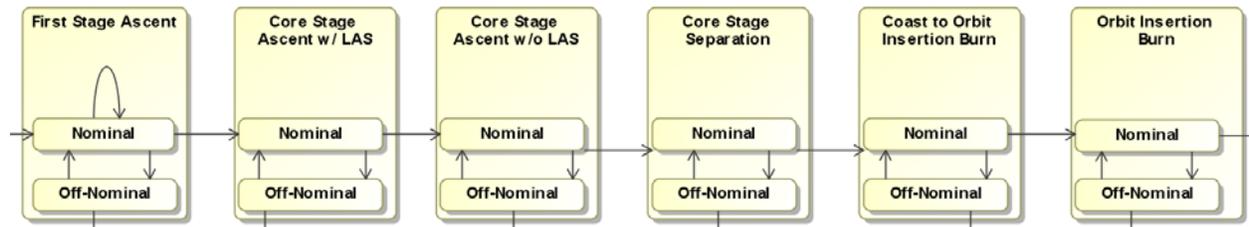


Figure 7: Launch Mission Phases (a portion of a larger diagram)

#### B. Physical Decomposition

The physical decomposition for this system begins with the SLS-MPCV Integrated Stack as the highest level component. The scope of this project mostly ignores the ground systems that would normal be involved, yet the system can still express critical behaviors of sending or receiving information to/from the ground by expressing the ground as an external entity – i.e., information goes out of the system scope and can be supplied into the integrated stack but no behaviors of the ground system are modeled. The integrated stack is then decomposed into the MPCV, the SLS, and an adapter section, which contains the Intermediate Cryogenic Propulsion System (iCPS).

Though many of the capabilities and requirements do not mention any subsystems directly, it can be established through some documented avionics information what many of these subsystems are. Furthermore, in later sections, functions can be attributed to these systems through general avionics knowledge. It is important to note that if this lower level of abstraction could not be expressed in the other viewpoints of the model, it would be useless to show within the physical decomposition and should not be detailed.

Many of the connections are determined from the IMA capabilities as well as information on the physical connectors found in the IRDs (e.g., telemetry flowing between SLS and MPCV via two redundant RS-422 interfaces). The physical decomposition is shown in Figure 8.

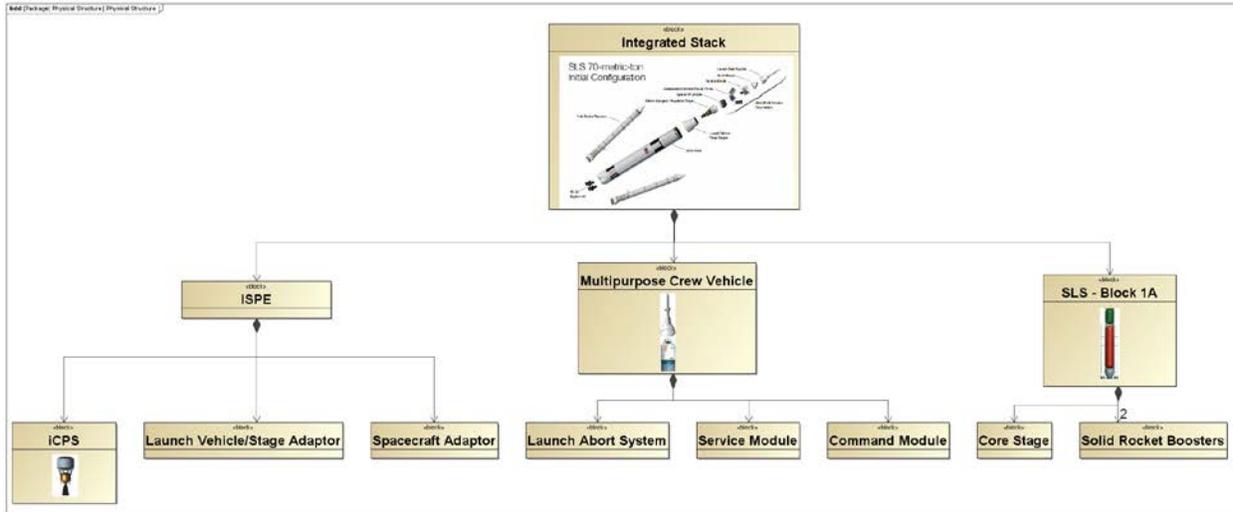


Figure 8: SLS-MPCV Integrated Stack Physical Decomposition

### C. State Analysis

To perform the state analysis for the integrated system, the capabilities were examined in depth. Many of these capabilities outlined the data that will flow between the SLS and MPCV systems, as well as where that data was generated. From this information, various states could be generated to show the generation, transformation, sending, receiving, and analysis of that data from end-to-end. However, many specifics of the data (e.g., what comprises specific data packets), were not mentioned in enough detail to model well.

To provide a brief example that is used within the model, a capability may outline the fact that SLS health and status data to MPCV. Therefore, it is known that the SLS has a health and status state variable, most likely composed of measurements from numerous sensors. The SLS compiles this information into a health and status message, which is transferred to the MPCV and analyzed. Another capability specifies that the MPCV can send an abort command to the SLS based on the SLS health and status. This capability shows the fact that there is a state command that is affected by the SLS health and status and can cause the SLS to abort – a path that affects numerous SLS states. The state effects diagram related to these two capabilities is shown in Figure 9.

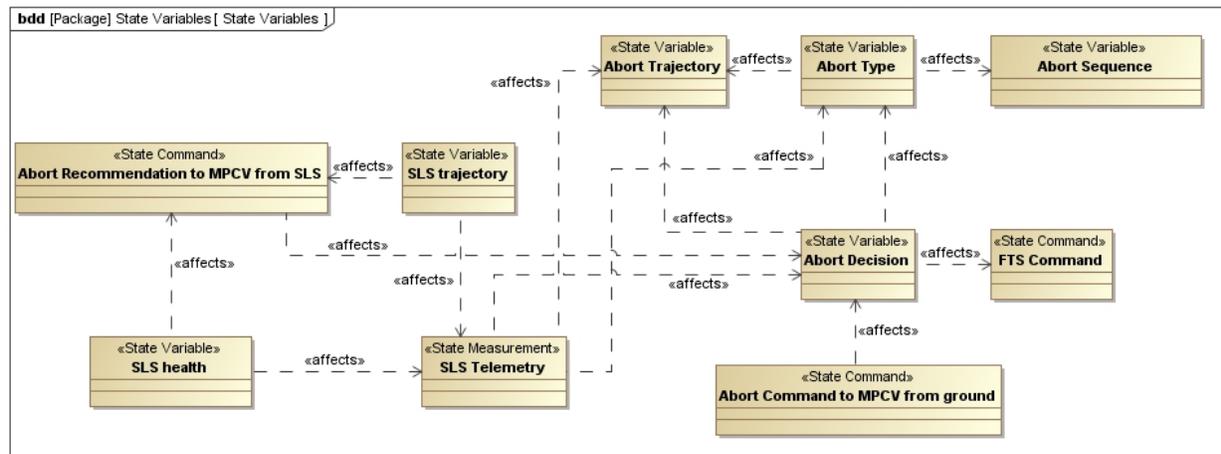


Figure 9: Applied State Effects Diagram

### D. Functional Definition

The functional definition can be derived from both the capabilities and the existing state analysis. For every “affects” relationship on the State Effects Diagram, there should exist a function that acts on those two states. Following the example from the last section, several functions can be outlined but only a few will be discussed in detail in order to remain brief. The two overall goals that are discussed in this paper are the SLS’s “Deliver Crew

and MPCV to Target Orbit Safely” and the MPCV’s “Perform Abort”. The functional decompositions for these two examples can be seen in Figure 10 and Figure 11.

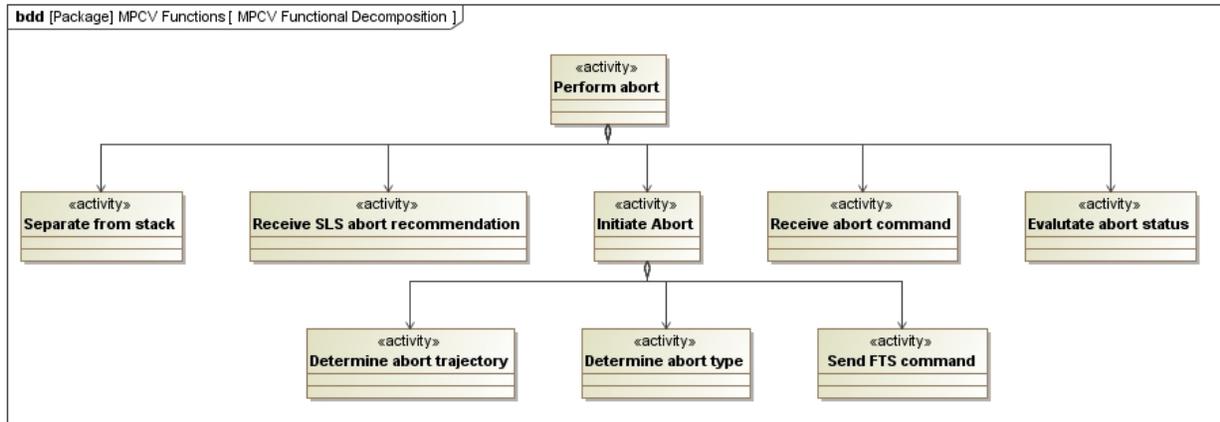


Figure 10: MPCV "Perform Abort" Decomposition

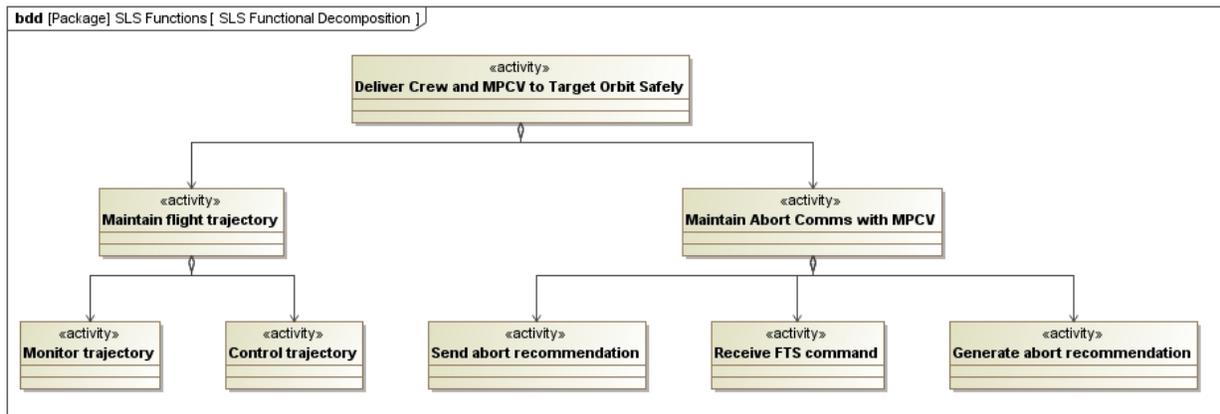


Figure 11: SLS's "Deliver Crew and MPCV to Target Orbit Safely" Functional Decomposition

Each of functions follows closely with some of the states that were previously defined. Specifically, the flight trajectory branch in the SLS tree utilizes the SLS trajectory and SLS telemetry states and could even be decomposed further by examining what states provide information on the trajectory and what measurements are taken to compose telemetry. However, for this example, the high-level approach will be shown. Both decompositions have a portion that discusses abort situations with some reciprocal functions (e.g., send/receive abort recommendation). This would be a case where an IRD or capability would be required to establish both sides of a communication link and if this could not be seen, a gap could be identified.

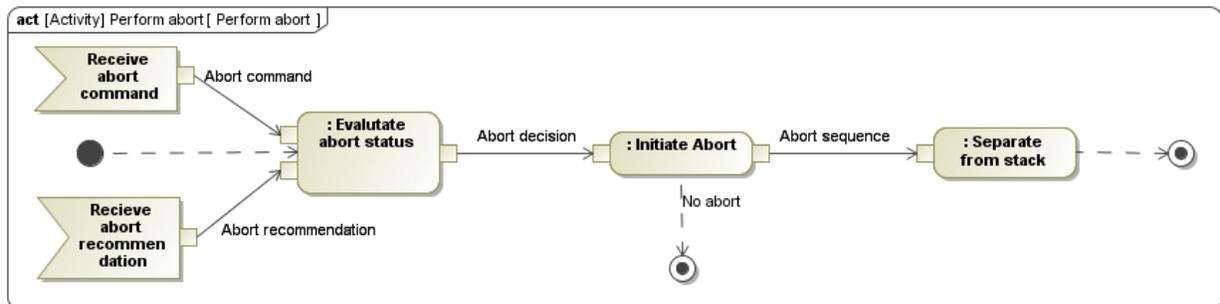


Figure 12: MPCV Functional Execution for "Perform Abort" Function

Figure 12 shows the functional execution for the “Perform Abort” function that waits for either an abort command from the ground or an abort recommendation from the SLS to be received, evaluates whether to abort, initiates the abort, and then physically separates from the stack. This functional execution shows a prime example of where different forms of analysis could be included. If this model were run as an executable (made possible by the Cameo Simulation Toolkit<sup>xiv</sup>), timing information could be included on each of the functions in this flow as well as their children functions. This timing could then show where certain abort situations may take too long to abort nominally, possibly causing loss of crew or loss of vehicle needlessly. These types of analysis can get more complex as the model evolves.

### E. Analysis

As previously discussed, a gap analysis can be performed on a completed model to ensure that all requirements are fulfilled, that each interface requirement has a reciprocal requirement on the opposite system (e.g. all send functions have a receive function and vice versa), and that anything specified by a capability has a coinciding requirement. Furthermore, gap analysis can be performed on the connections outlined by requirements or capabilities to ensure that one end of the connection has the same constraints and properties as the other side (e.g., both use the same type of connection). This connectivity analysis is automatically performed in MagicDraw<sup>vii</sup>. An example of performing a gap analysis on an IRD can be seen using the *ibd* in Figure 13. Here the two reciprocal IRDs would be linked to the sending and receiving functions, the ports on the systems, and the physical connector between them.

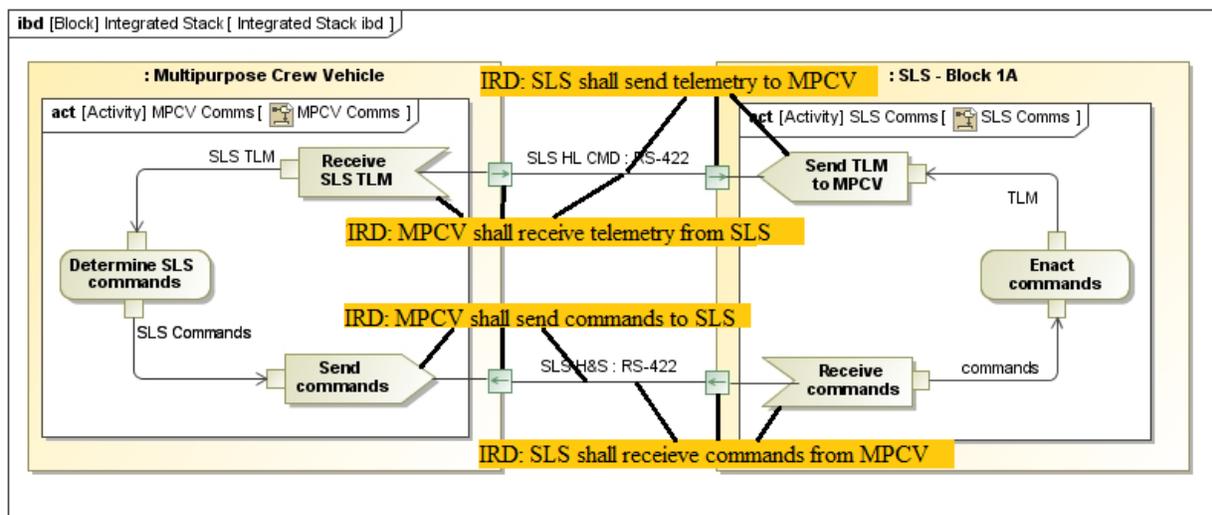


Figure 13: An *ibd* Showing the Capability for Requirements Gap Analysis

Though our focus has mostly been on the analysis of gaps within the model, many other forms of analysis are possible. As mentioned before, the model is built in a fashion that allows it to become an executable program. To properly do this takes more time than was allotted for this task. With an executable model, any analysis of properties of the system could be performed as far as the proper level of detail was provided in the model.

## IV. Conclusion

By developing a unique methodology described in this paper for behavior modeling, the SLS-MPCV integrated stack is able to be modeled and analyzed to answer questions (i.e., address concerns) that could not be otherwise easily addressed. The process has been performed on several facets of the systems, examining flow of telemetry between subcomponents of the SLS core stage as well as the abort situations between MPCV and SLS that were used as an example earlier in this paper. Though not every aspect of the model could be discussed in this paper, the core takeaway from this effort is the selected process. By combining MBSE and state analysis, this process allows for detailed control theory-based behavior analysis within a cross-system model.

There are many possibilities to continue the work defined in this paper. Despite the author’s departure (at the end of his internship), the task is being continued by the rest of the modeling team at JPL as well as advisors at other NASA centers. Primarily, the process will continue to be applied to the SLS-MPCV single engine out scenario to capture the behavior of the integrated stack during aborts. Similarly, continuing to perform a gap analysis within the

model until all IRDs are verified is a must. After having the system fully modeled to a desired level of abstraction, more analysis methods can be used to make full use of the information contained in the model. Furthermore, an executable model could be created from the model, allowing the time-based analysis of any state variable. Defining the process to create this model was a large portion of work, but it only opens the door to better capabilities for analysis.

### Acknowledgements

I would like to thank Oleg Sindiy, Oti Liepack, Jessica Clark, Kim Simpson, Thom McVittie, Mitch Ingham, Dan Dvorak, all of the JPLers who helped answer my modeling questions, and many others who provided advice and guidance during my time spent on this project. This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the JPL Student Internship Program and the National Aeronautics and Space Administration.

### References

- 
- <sup>i</sup> “Apollo 17,” NASA, [http://www.nasa.gov/mission\\_pages/apollo/missions/apollo17.html](http://www.nasa.gov/mission_pages/apollo/missions/apollo17.html)
- <sup>ii</sup> “Low Earth Orbit,” NASA, [http://www.nasa.gov/audience/foreducators/k-4/features/LEO\\_index.html](http://www.nasa.gov/audience/foreducators/k-4/features/LEO_index.html)
- <sup>iii</sup> “Beyond Earth: Space Launch System,” NASA, <http://www.nasa.gov/exploration/systems/sls/>
- <sup>iv</sup> “Orion,” NASA, <http://www.nasa.gov/exploration/systems/mpcv/index.html>
- <sup>v</sup> Aguilar, M., *Modeling and Simulation of System Behavior at SLS, MPCV, and GSDO Interfaces*, NESC-NASA, Internal Document.
- <sup>vi</sup> *OMG Systems Modeling Language*, Objects Modeling Group, <http://www.omgsysml.org/>.
- <sup>vii</sup> *MagicDraw*, No Magic Inc., <http://www.nomagic.com/products/magicdraw.html>
- <sup>viii</sup> Estefan, J. A., “Survey of Model-Based Systems Engineering (MBSE) Methodologies,” INCOSE MBSE Initiative, May 23, 2008.
- <sup>ix</sup> Ingham, M.D., et al., “Engineering Complex Embedded Systems with State Analysis and the Mission Data System,” AIAA.
- <sup>x</sup> Johnson, S. B., “Goal Tree/Success Tree Modeling: Presentation to NESC Behavioral Modeling Team,” Internal Document, June 12, 2012.
- <sup>xi</sup> *MagicDraw: Tutorials*, No Magic Inc., Ver 17.0, 2011, <http://www.magicdraw.com/files/manuals/MagicDraw%20Tutorials.pdf>
- <sup>xii</sup> *Exploration Systems Development Integrated Mission Analysis Report*, NASA-ESD 10014, Draft, Feb 29, 2012.
- <sup>xiii</sup> “Exploration Systems Development Division (ESD), *Independent Program Assessment Office*, NASA, 2012.
- <sup>xiv</sup> “Cameo Simulation Toolkit,” No Magic Inc., <http://www.nomagic.com/products/magicdraw-addons/cameo-simulation-toolkit.html>