Creation of a SysML Model to Aid Design of an Autonomous Soldier Supply Vehicle

Ryan Arlitt
Systems Engineering 401
5-11-10

Table of Contents

Abstract

A partial SysML model was created for an autonomous soldier supply vehicle. The model was created in parallel with document-centric system development. The modeling approach rebuilt elements from the document-centric mode within the Artisan Studio SysML environment, then further synthesized several areas of the design within the model-centric environment. Particular attention was paid to the creation of activity diagrams to drive the design of system structure. Conclusions are drawn about the successes and failures of the approach, as well as the usefulness of specific model types within Artisan Studio.

Model

System Domain

The vehicle domain diagram lays out the operational environment of the system. Modeling the system domain gives context to the model, allowing traceability to be shown to the system's various external interfaces. Actor icons represent external systems or humans that will interact with the system. Blocks under the MARMOT Domain block represent components of the system domain. At this level, elements are still relatively abstract. For example, “sand” is modeled here abstractly as a component of the physical environment. The goal of this high level diagram was to simply identify as many interfaces as possible.

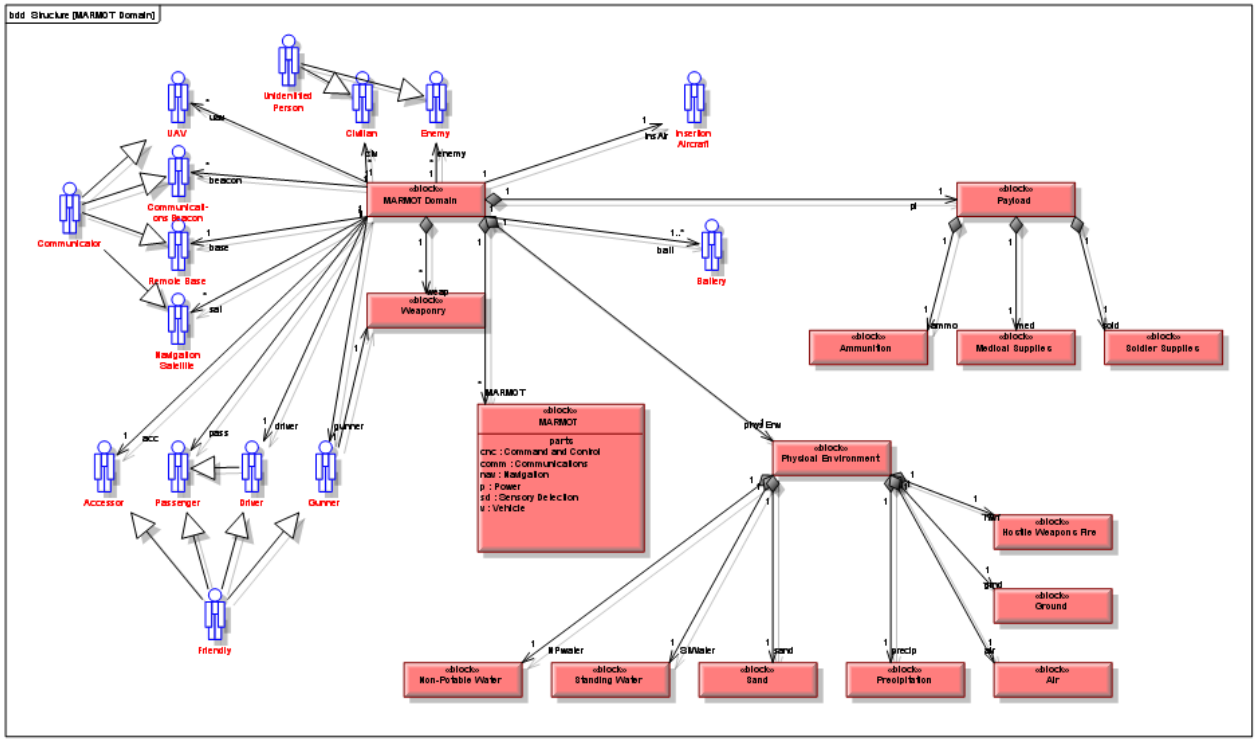
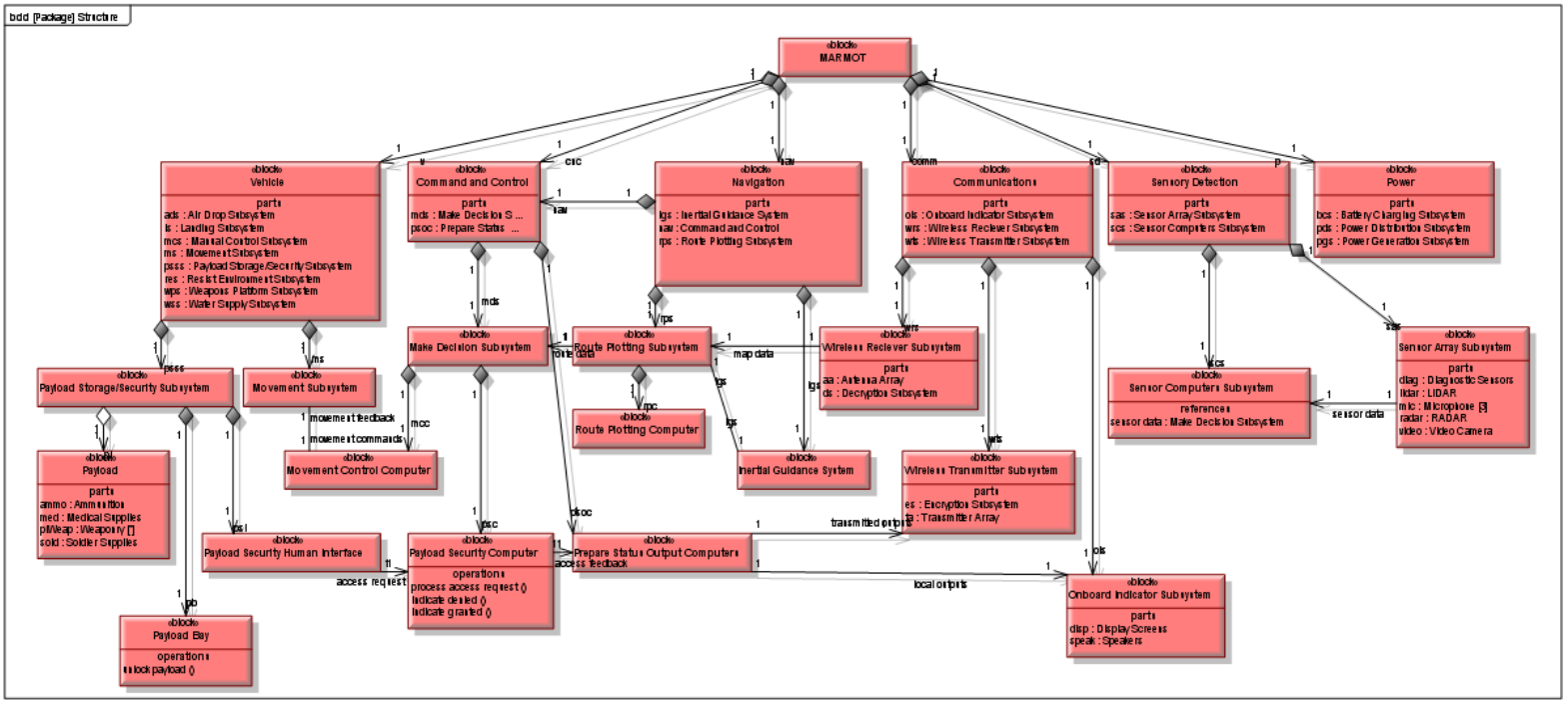


Figure 1: System Domain

Architecture Hierarchy

The architecture hierarchy lays out the system architecture in terms of the project structure. The six primary subsystems used to devise the project's work breakdown structure further break into specific task-oriented subsystems. These subsystems were derived directly from the required capabilities devised during document-centric development. These subsystems can be seen in the "parts" compartment of each primary subsystem block. Further hierarchical breakdown shows subsystem components and important interfaces between those subsystems. For example, Command and Control's Make Decision Subsystem, which satisfies part of the Control Movement capability in the functional analysis, breaks down here into the Movement Control Computer and Payload Security Computer. The Make Decision Subsystem has interfaces to the Route Plotting Subsystem and the Sensor Computers Subsystem (shown here by the "references compartment"). The Movement Control Computer interfaces with the Movement Subsystem via movement commands sent and movement feedback received. Meanwhile, the Payload Security Computer receives inputs from the Payload Security Human Interface and outputs to the Prepare Status Outputs Computers. These connections reveal interfaces with both the Vehicle and Communications primary subsystems. Similar architecture breakdown and interface data can be found throughout Figure 2: System Hierarchy.

Figure 2: Architecture Hierarchy



Power Distribution Structure

The internal block diagram in Figure 3: Power Distribution Structure shows how the power subsystem interacts with the five other subsystems. Untyped flow ports model the data transfer to and from the power subsystem. For example, the sensory detection subsystem requests energy from the power subsystem, then the power subsystem delivers feedback on the power available for delivery. The unidirectional flow port models the actual transfer of energy. Each main subsystem interacts with the power subsystem in this manner.

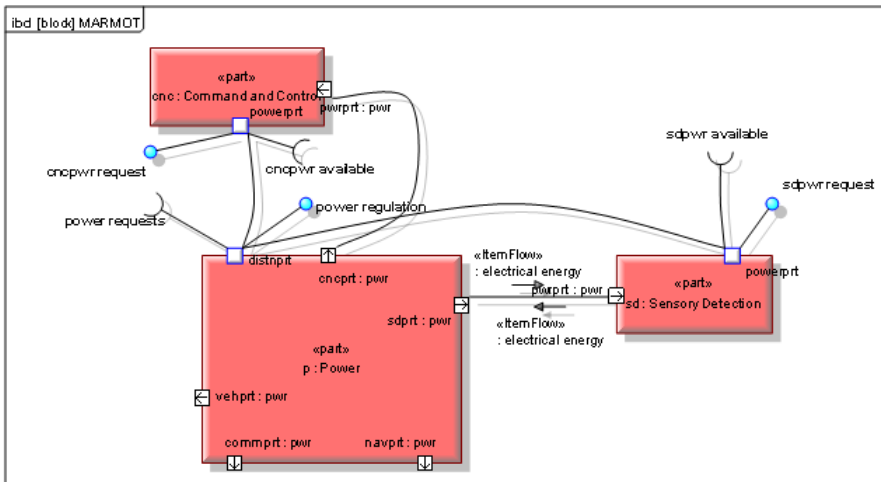


Figure 3: Power Distribution Structure

Operational Use Cases Involving Personnel

A use case diagram, seen in Figure 4: Operational Use Cases, was developed to organize how various types of personnel interact with the system. These use cases would each provide guidance for system capabilities requiring development. The “request payload access” use case was further developed in a sequence diagram that led to the discovery of components needed to satisfy the capabilities derived from the use case.

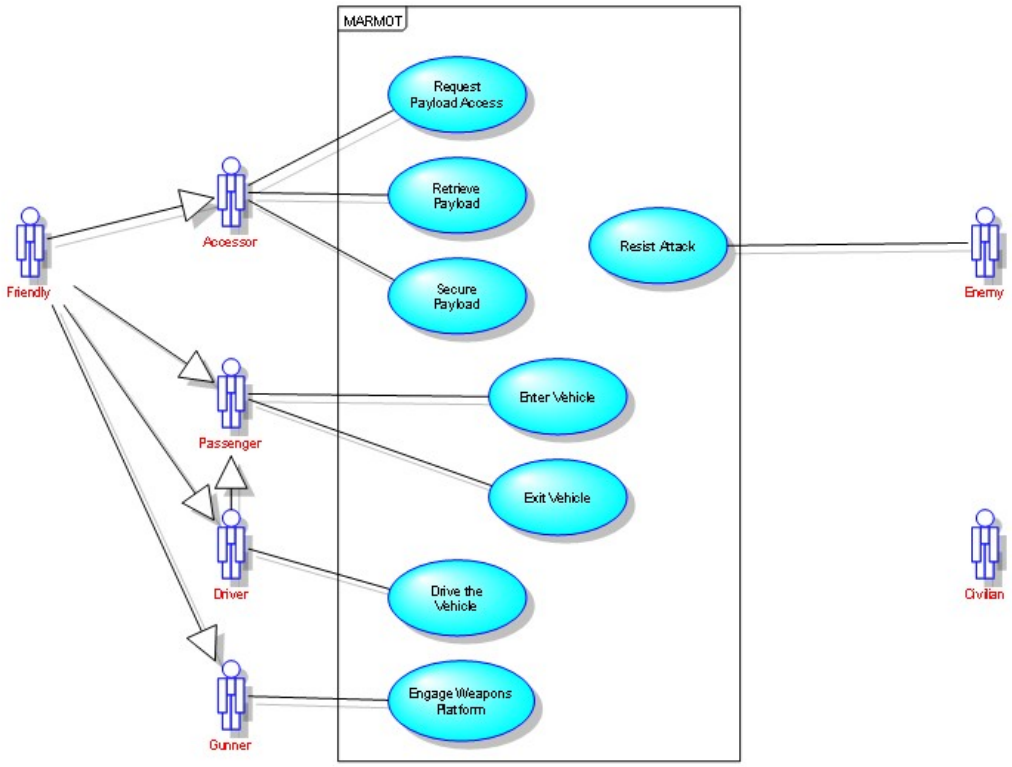


Figure 4: Operational Use Cases

Operational States

The state machine diagram shows the high level operational states of the MARMOT. The system begins with the startup state, then checks for orders. In the absence of orders, the system enters a hibernating state. Otherwise, the system begins normal operation until it completes its orders. In the event that the system detects a threat, it enters an alerted state wherein it continues to perform its mission, while carrying out appropriate defensive maneuvers, until the threat disappears. While hibernating, the system waits for mission orders or a shutdown command. The "perform mission" state has traceability to the perform mission activity seen in Figure 7.

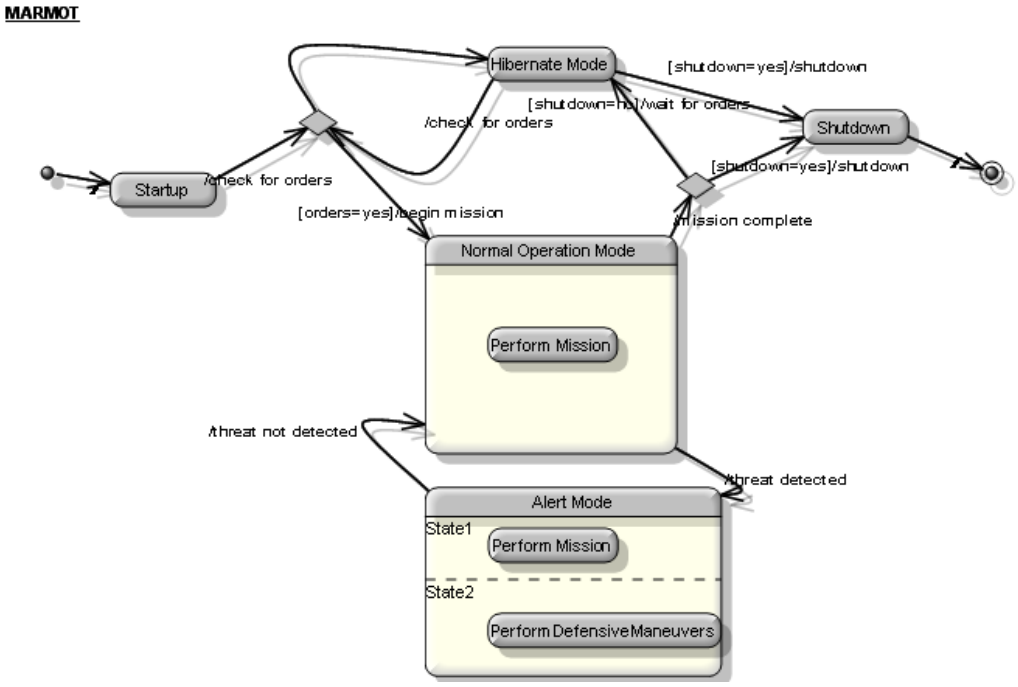


Figure 5: Operational States

Top Level Function Flow Block Diagram

Functional decomposition began in the document-centric domain with the creation of a top level function flow block diagram (FFBD) seen in Figure 6. The diagram outlines high level call behavior activities required of the system, and flows down into lower level capabilities. Figure 7 represents the conversion of the FFBD into a SysML activity diagram. The translation into SysML was not directly useful at this level, but it allowed decomposition into more useful lower level activities.

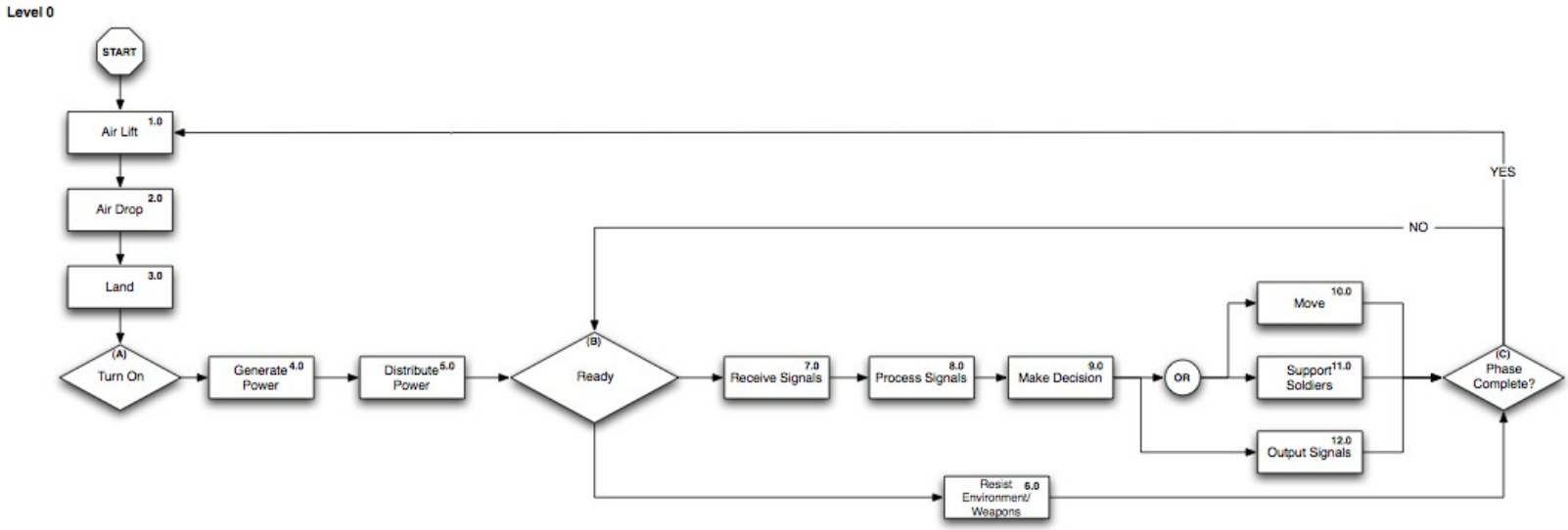


Figure 6: Top Level FFBD

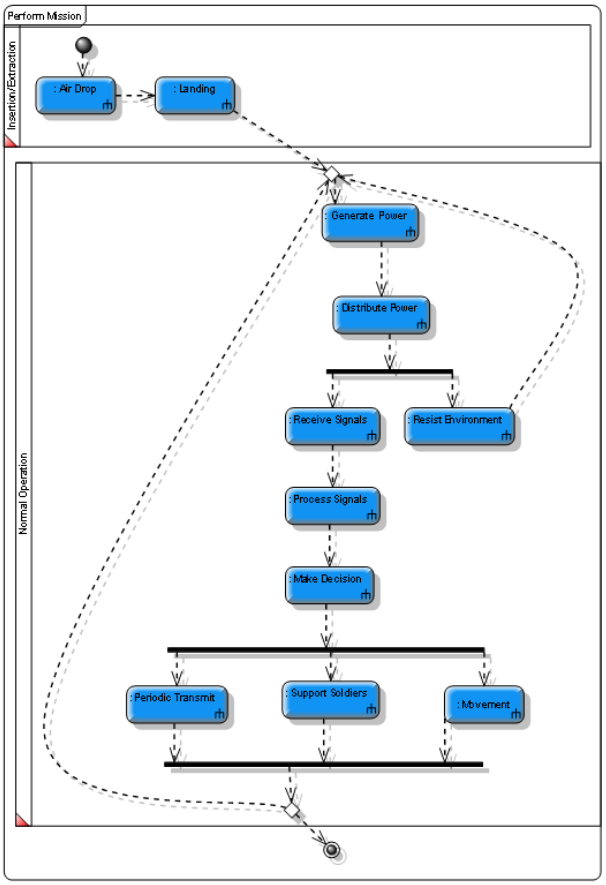


Figure 7: Perform Mission Activity Diagram

Support Soldiers Activity

The support soldiers activity within the perform mission activity broke down into the four capabilities seen in Figure 8: Support Soldiers Capability. Materials, energies, and actors were all treated as flows through each activity to better preserve the meaning of the original functional diagram. Decomposition occurred iteratively as each activity broke down into smaller black boxes; each block is defined in terms of its inputs and outputs without regard for what happens inside. The support soldiers activity broke down into the capability to recharge portable devices, the capability to serve as a weapons platform, the capability to store and secure payload, and the capability to supply drinkable water.

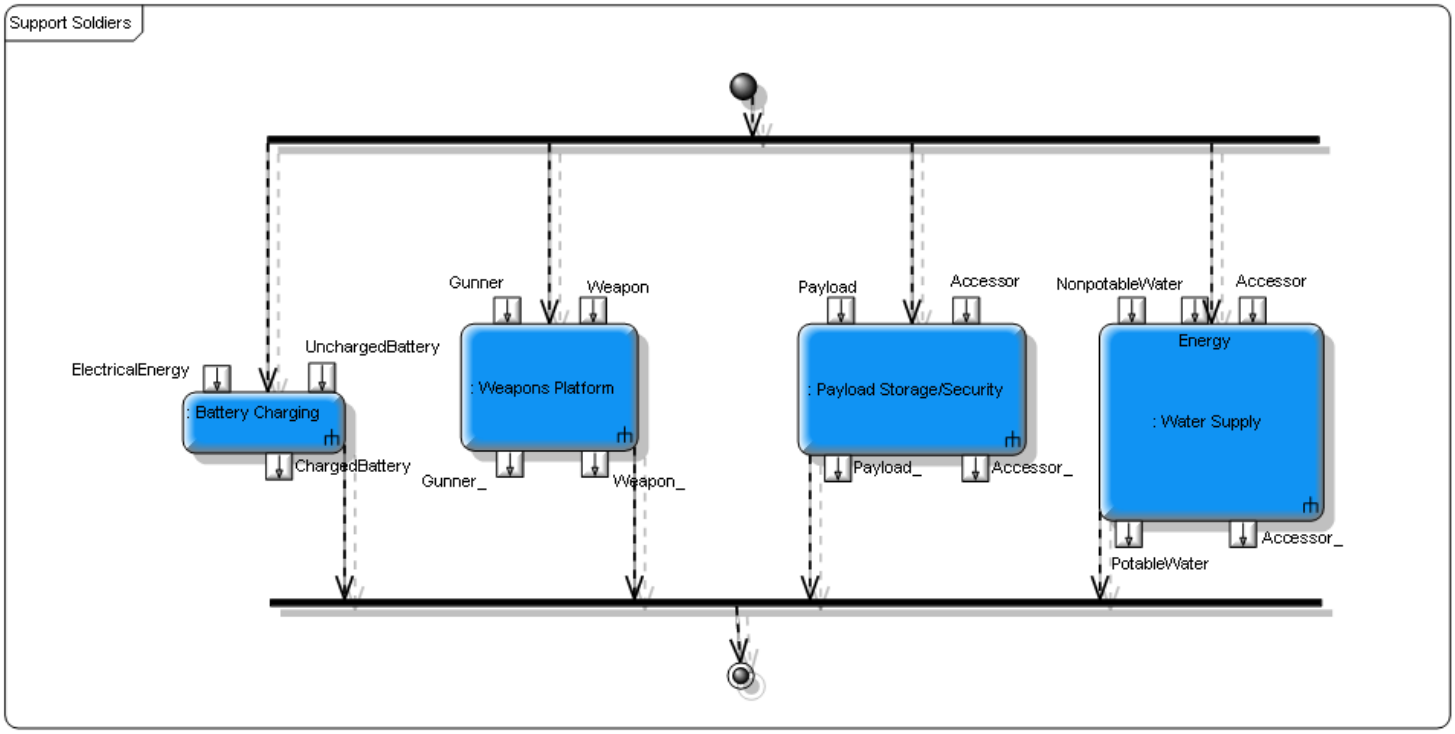


Figure 8: Support Soldiers Capability

Battery Charging Capability

The battery charging capability from the support soldiers activity broke down further into the activity diagram shown in Figure 10. The document-centric decomposition of the battery recharging capability (Figure 9) was developed independently. The SysML representation preserves the meaning of the functional model document while providing the additional integration to the rest of the model. The requirements mapping attempted in Figure 9 plays into a natural strength of SysML. Links can be created between model elements rather than simply writing requirements identifiers next to function blocks. These low level functions are also the first example of the functional basis terminology used in the model. This functional basis terminology [1] was developed to encompass all possible functions performed by electromechanical products. This allowed a high level of solution independence when determining the functions of the system. In this case, the model depicts an uncharged battery entering the system, being secured in place, being charged, and then leaving the system.

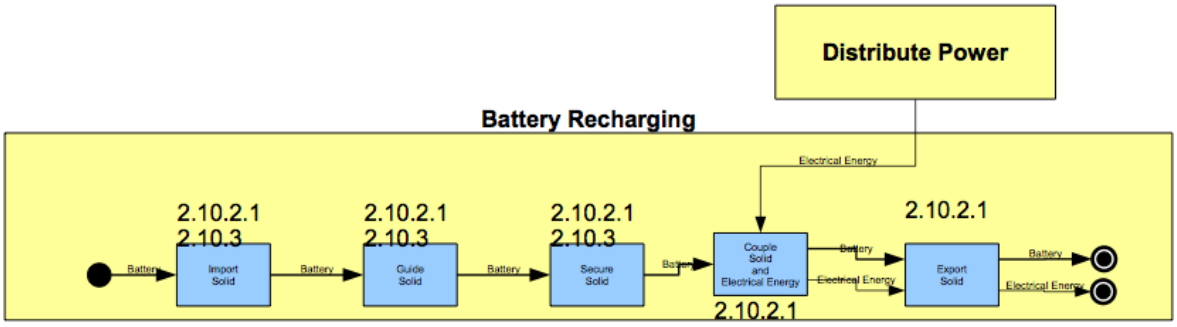


Figure 9: Battery Charging Capability

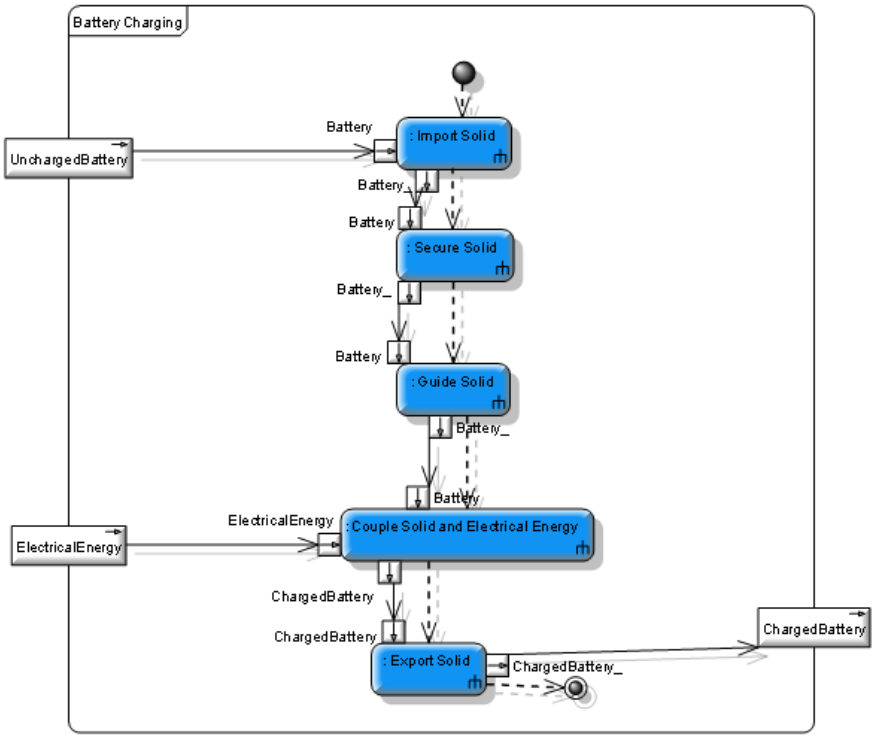


Figure 10: Battery Charging Activity

Receive Signals Activity

In another example of functional breakdown, the receive signals activity from the high level perform mission activity was decomposed. The activity diagram depicts the flows into and out of each call behavior activity, along with traceability to use cases and architecture elements. The “manual control” activity maps to the “drive the vehicle” use case and the “manual control” subsystem. The “communication in” capability maps to the “wireless receiver” subsystem. The “sensing” capability maps to the “sensors” subsystem. Flows through each activity again show the important inputs and outputs to each capability.

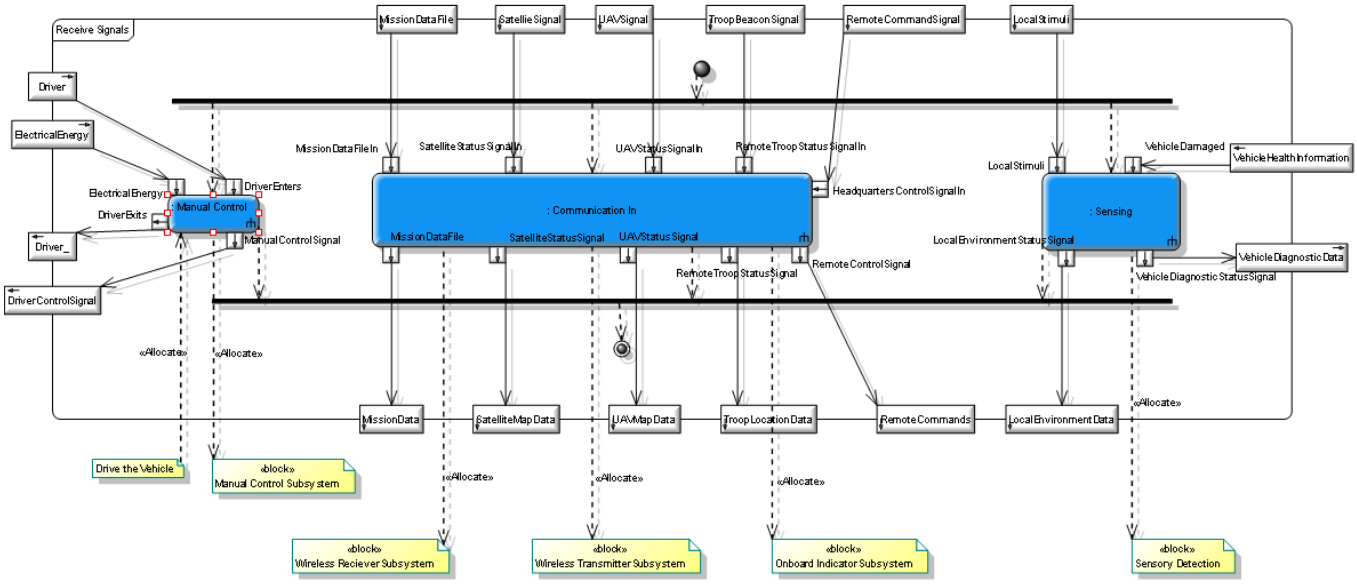


Figure 11: Receive Signals Activity

Manual Control Capability

The manual control activity (Figure 12) depicts the flow of the driver and electrical energy through the system, eventually outputting the driver and a control signal. After the driver flows into the system, he is positioned and stored in the control area. From there, the system interacts with the driver's hands and feet to receive command inputs. This occurs via a combination of driver actions and electrical energy to convey the signal. Those signals then exit the activity and would enter a new activity that models the vehicle's movement actuators. Eventually the driver exits the system, ending the activity until a new driver is present.

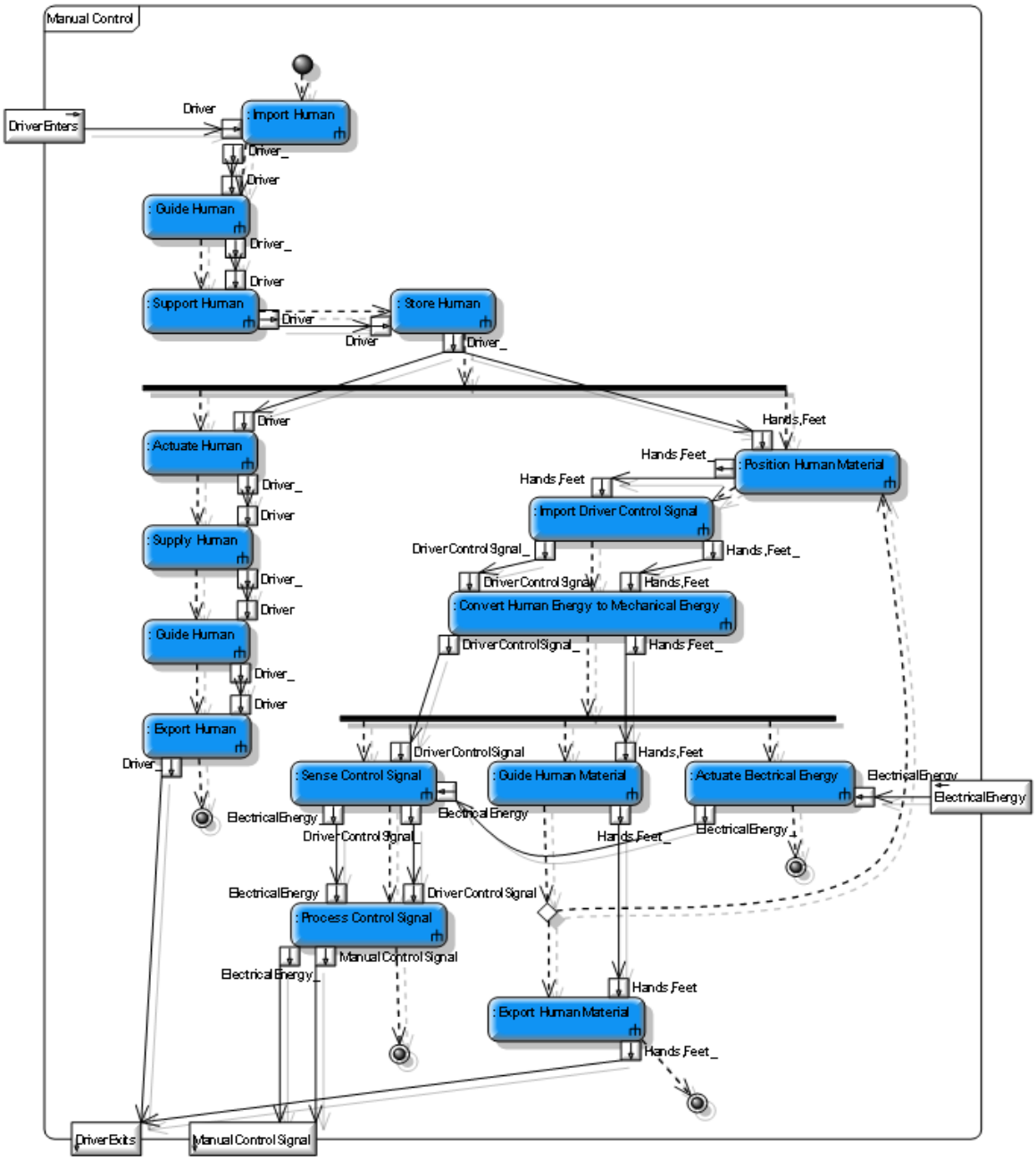


Figure 12: Manual Control Activity

Access Payload Sequence Behavior

Figure 14: Payload Access Behavior shows timing behavior of the Payload Storage/Security Subsystem when an access request is made. The accessor relays an access request event to the Payload Security Human Interface, which then relays the request to the Payload Security Computer for processing. If the access request matches the correct request code, the computer relays an access granted message to the human interface while simultaneously unlocking the payload bay. At that point the accessor is free to retrieve payload. If the code does not match, the interface indicates access denied. The final event depicts

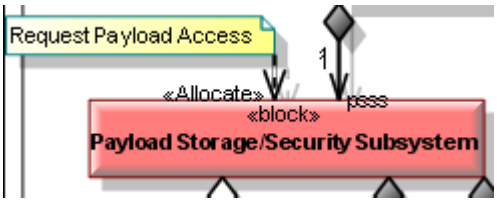


Figure 13: Payload Storage Traceability

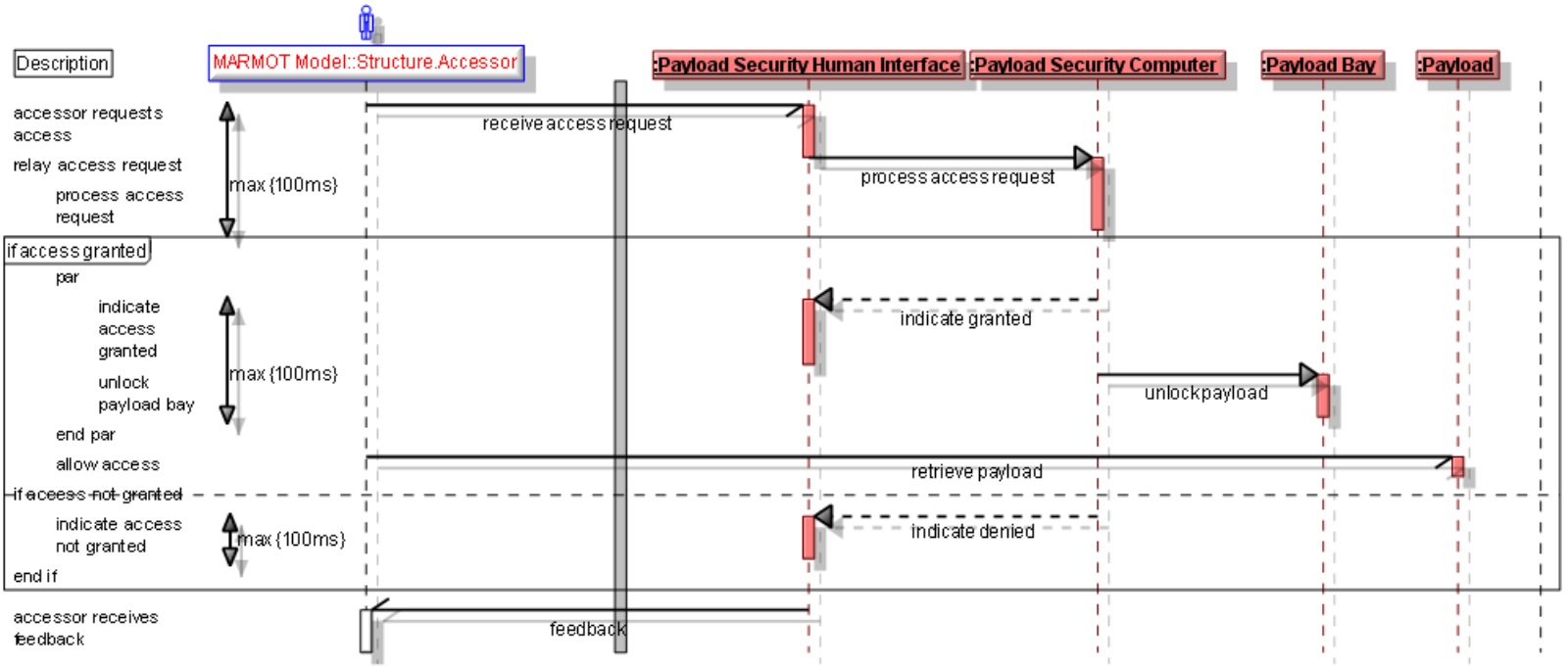


Figure 14: Payload Access Behavior

Requirements Traceability

Figure 15 shows an example of requirements traceability applied to the Payload Storage/Security Subsystem. The subsystem satisfies 2.5.3.1 Locking Mechanism and 2.5.3.2 Lock Human Interface. The Locking Mechanism requirement contains 2.5.3.1.1 Allow Access and 2.5.3.1.2 Deny Access. Due to the cumbersome nature of requirements diagrams, only a small sample was created.

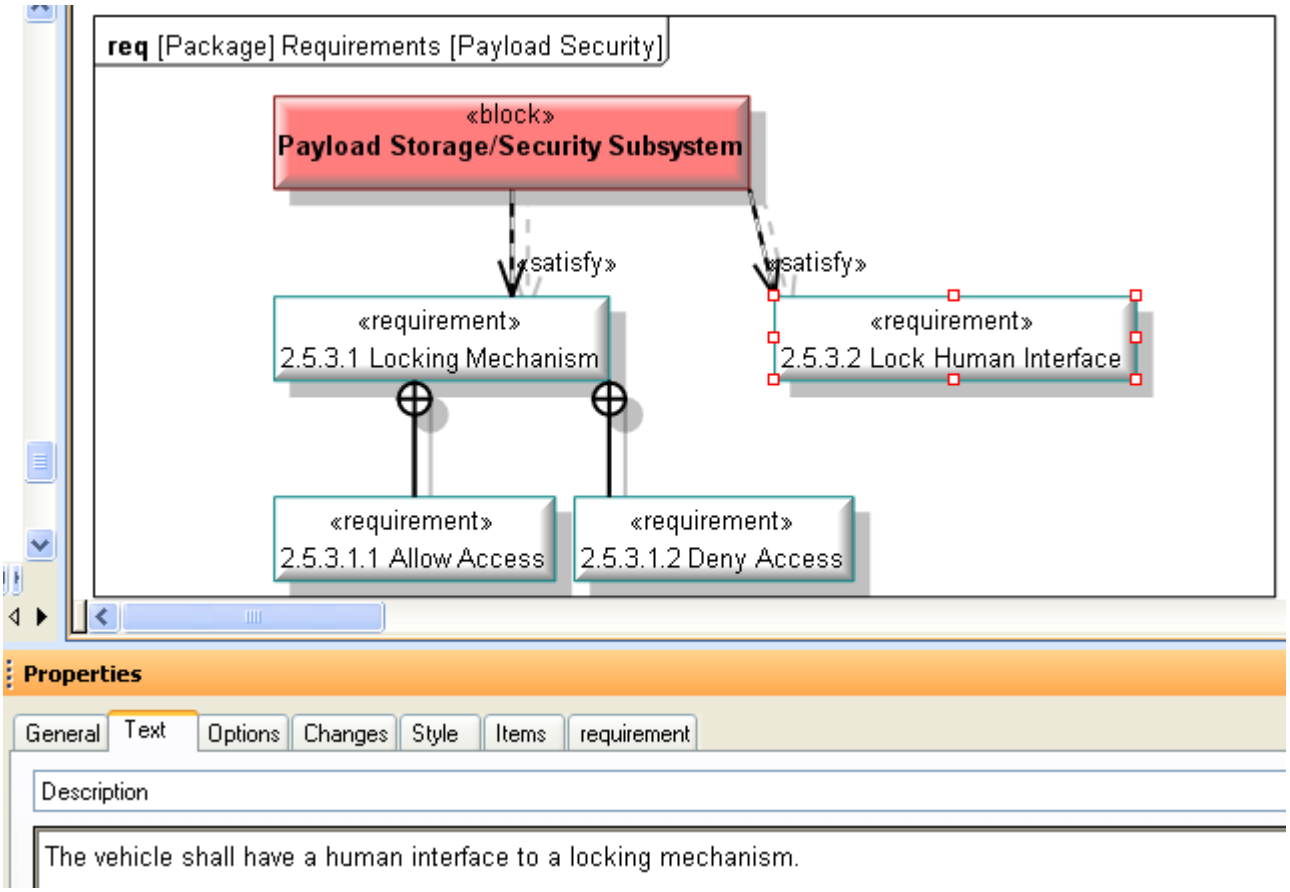


Figure 15: Requirements Traceability Sample

Conclusions

Activity diagrams provided clear traceability from required capabilities to components. Modeling the required system capabilities using activity diagrams aids in the identification of functionally critical flows through the system. This will provide an additional benefit later by highlighting the distinction between component capabilities that are critical and those that appear to be needed to support the critical functionality.

Due to resource constraints and the learning goals associated with the creation of the model, attention was focused on certain model sections. The systems engineering process was forgone in favor of gaining experience with a variety of SysML aspects. A proper full design within SysML should begin with the creation of requirements diagrams based upon customer needs. Clear traceability could then be shown from requirements, to functions, to components. This would have dramatically improved the clarity of the model. In this case, poor integration with DOORS and the clumsy nature of requirements diagrams necessitated that model creation efforts be focused elsewhere.

Further model synthesis is required to create a true representation of the system. Activity diagrams must be filled in for each of the capabilities that flow down from the high level function flow block diagram. Analysis must then be performed on the critical low level activities to determine the types of parameters involved. From there, block definition diagrams can be constructed concerning the components that satisfy those functions.

Acknowledgments

I would like to thank Daniel Bittner, Darren Charette, Robert Davey, and Kristen Donovan for their work in document-centric development of the system.

References

1. Hirtz, Julie, Robert B. Stone, Daniel A. McAdams, Simon Szykman, and Kristin L. Wood. "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts." *Journal of Research in Engineering Design*.