2014 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING (SE) TECHNICAL SESSION
AUGUST 12-14, 2014 – NOVI, MICHIGAN

MODEL BASED SYSTEMS ENGINEERING USING VALIDATED EXECUTABLE SPECIFICATIONS AS AN ENABLER FOR COST AND RISK REDUCTION

George Fortney
SAIC.
Sterling Heights, MI

ABSTRACT

Current written system specifications have a high degree of uncertainty which causes specifications to be changed because they are incorrect, incomplete or do not possess the degree of rigor to make them precise. Even when generated by modeling methods such as UML/SysML or standards such as DoDAF, these functional specifications still lack any validation with respect to architecture, mission, and scenario impacts. The lack of consideration of these aspects creates design errors are usually exposed during the test and integration phases where the expense is greater to correct than in the early conceptual design phase. This paper will introduce the concept of Validated Executable Specifications (VES) that will enable Model Based Systems Engineering (MBSE) to validate early in the design process to reduce risk and save costs in a System of System (SoS) model.

INTRODUCTION

Tests have shown that 65% of all design faults occur on the left side of the Systems Engineering Vee, but only 3.5% are found.[1] This is where the costs to correct are cheapest and quickest. As we move up the right side of the Systems Engineering Vee, fewer faults are introduced and more are discovered. During Integration and Test, only 10% of new faults are introduced and 50.5% are found. The costs to remove these faults are a factor of 10 higher compared to correction during the concept design phases. The lack of early fault identification and remediation results in cost overruns and increased risk. The greater the system and its complexity, the greater the cost and schedule overruns and overall risk.

When coupled with a virtual test environment, the use of Validated Executable Specifications (VES) can be enabled. Model Based Systems Engineering (MBSE) models using VES can validate models in the concept design phase significantly reducing faults in the test and integration phase. MBSE with VES in the concept design phase will discover more faults and offer the ability to correct them when the cost to do so is lowest. A VES model can be executed and coupled with other functions, architecture elements, and resources to develop a complete mission level System of System model. This type of model allows one to run what-if scenarios for optimization, augmentation of the system, changes in component parts or to test capacity and robustness. This paper will discuss concepts like UML and SysML and why they do not offer the capability that MBSE with VES bring. MBSE with VES enables resource availability and optimization modeling, discrete event (DE) simulation with time correlation, both synchronous and asynchronous data and signal flow, finite state machine (FSM) for control flow, and continuous time modeling for analog effects. This paper will discuss how existing UML and SysML models can be used as a basis to introduce a MBSE virtual test environment as the framework for true VES to provide 100% reuse of the current architectures.

There are many commercial and military use cases where MBSE with VES have been used to deliver on-time and on-budget project results. There are also many use cases where MBSE and VES were successfully used for root cause analysis in both systems that were designed via MBSE and those that were not. A good cross section of various examples of these use cases will be discussed to show how they overcame the impediments for success, how they reduced costs and risk as well as overall lifecycle costs.

The paper will then discuss how MBSE and VES can be used to create System of System (SoS) models for ground vehicles that will be VICTORY architecture enabled. This is a perfect methodology for reducing risk and costs when integrating VICTORY enabling technology. The paper will discuss lab based instantiations using MBSE with VES to design and optimize integration and as a proxy for data input for inter-vehicular communication. The ability of MBSE with VES to reduce the time to production due early fault detection and correction will also be discussed. This will reduce the costs and risks of traditional validation during the

integration and test phase. The value of the application of an executable integrated SoS model in integration labs, Hardware-in-the-Loop (HITL) testing, Golden Benches and for Lab Based Risk Reduction (LBRR) will be discussed.

VES in the System Engineering Vee model

Unlike the traditional role of validation occurring in the Integration & Test right hand side of the VEE. MBSE validates on the left hand side where errors are quickly found and corrected with less costly effort. Different studies indicate that most design errors are introduced in the very first levels of systems design, during concept and specification phases. Most errors are discovered in the development process predominantly during integration or even when the product is deployed. Estimates for the amount of errors introduced during specification phase range between 60% and 70%. Figure 1 shows respective values for complex software development. It is derived from the National Institute of Standards & Technology (NIST) Planning Report 02-3 in 2002.

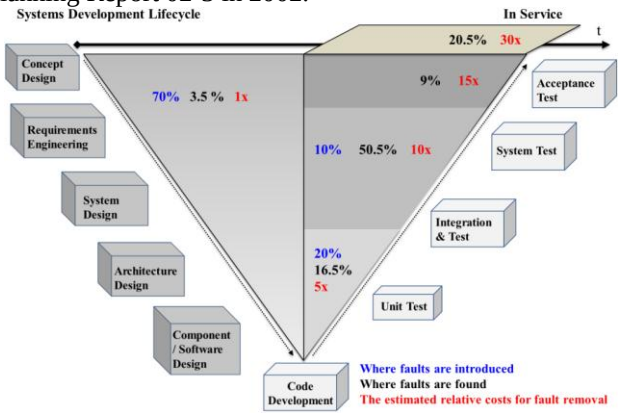

Figure 1: V-Model including percentages of where errors are likely to be introduced and found together with estimates of relative costs for error removal within the systems development lifecycle, adapted from [2]

In the current development process functional specifications are developed and distributed to Software (SW) developers. These developers are developing the SW without knowledge on how the data gets to the SW from the outside and connect the different SW elements purely functionally. Even for this development stage they typically spent more than 80% of the development time on data transport in their functional level SW. This SW development does not consider

- Hardware (HW) limitations
- Different arrival times for data
- Common resources used by many different applications, like busses, buffers

During integration in HW everyone is struggling to get their data to their subsystems so during Integration a major redesign starts. Since it is late in the development cycle these redesigns are done at high cost and pressure to get things completed. Typical projects experience major cost and time overruns primarily because of this reason. Developing during concept development (also for redesigns during life cycle management) a common data transport model together with a resource models of HW and protocols specification tradeoff and validation and following this stage by integrating functions elements for a virtual prototype will avoid most of the problems.

Examples:

- One of the most complex airplanes, the NAVY P-8A Poseidon, was produced with no critical errors and experienced no delays – an aviation first for such a sophisticated platform. VES were used from concept development through life cycle management. The contractor's choice to adhere to executable specifications created a digital thread under which the sub contractor supply chain had access to the SoS model and as a result the performance, cost, power, etc impact from any and all coupled effects that engineering changes may cause were visible and thus validated against the overall specifications. Since the SoS model utilized mission and scenario parameterized models, NAVAIR was able to further sanction the models for mission training purposes.
- Development of the SW for the ESA standard for communicating to Satellite subsystems purely on a functional level with UML took 9 months and did not work. Developing the SW together with a data transport model and test at application/mission level of communications between satellite and ground station took 1 week. The second approach avoided all problems which the 1st approach had.

MBSE and the limitations of UML/SysML

UML and SysML are concepts for functional modeling that programs such as MagicDraw® and Rational Rhapsody® use to visualize, manage, and organize functionally and activity/program flow but do not fully address architectural impact, dynamic networked feedback loops, and other resource sensitive aspects. UML and SysML are primarily used for SW development in devices. Airbus looked at UML/SysML and concluded it was too limited in scope for anything other than small and simple projects. It does not impact the risk associated with anything substantial because it does not provide for an integrated development environment. To accurately model resource availability you need a multi-domain simulation environment that includes discrete event simulation which UML does not address. You also need time correlation which is event based to know when the resources are free, the quantity shared as well as server shared resources to know that capacity of resources and how long things take to process tasks. With UML you can model limited functional

behavior, e.g. state chart behavior. However, you cannot model availability of resources. Therefore you cannot model architecture and cannot validate. With SysML/UML you add some simulation capability to UML. However, you cannot model architecture together with functional behavior. You therefore cannot validate. Validated Executable Specifications (VES) that take architectural models and add functional modelling capabilities can validate function, architecture and integrated architectural/functional behavior at mission/application level. With UML and UML/SysML you start the validation process when you integrate. You therefore have to correct errors when the cost is higher. This typically causes large cost overruns of projects. With MBSE and VES you validate early in the design when cost to correct is lower. This significantly reduces the risk of a project.

This is not to say that UML and SysML are not without merit. They do what they are designed to do well and can be utilized in an MBSE construct. In order to add the MBSE functionality to UML/SysML those models need to be bought into an MBSE virtual test environment. They then become the framework to add the logic to for the creation of VES. All of the hard work that went into these UML/SysML architectures has 100% reuse in an MBSE environment. The MBSE virtual test environment can be used to start the validation process for these UML/SysML models.

SAIC did a proof of concept porting UML into a MBSE framework for PMO SBCT Modernization, Stryker Vehicle Electronics [3]. We were given a StarUML XMI file to import into a MBSE tool (in this case Mission Level Design –MLD). One of the first issues we encountered was the inherent limitations of UML version 1.3 gave structural data only as XMI 1.x which does not provide presentation data so there was no visual representation. The UML project contained ActivityGraph (ActivityGraph1) \with a single StateMachine that contained

16 UML:ActionState,
2 UML:Pseudostate,
3 UML:FinalState
24 UML:Transition

We created a Finite State Machine (FSM) as seen below:

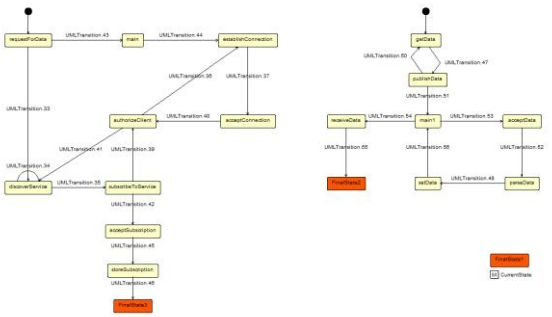

Figure 2: MBSE FSM created from a UML 1.3 model created with StarUML 5.0.2.

Some of issues we discovered in doing this proof of concept were that FSM Design Top Level contains more than one Default Entrance, Synchronous Self-Transition without Guard Condition of State: the 'discoverService' causes an infinite loop, State: 'FinalState1' cannot be reached by at least one non-self Transition, FinalState 1 through 3 cannot be left. It was determined that more information would be needed in order to do a simulation. It was also determined that UML 2 with additional modeling tool and guidelines will achieve an executable simulatable model. From this FSM we created a client and server model.

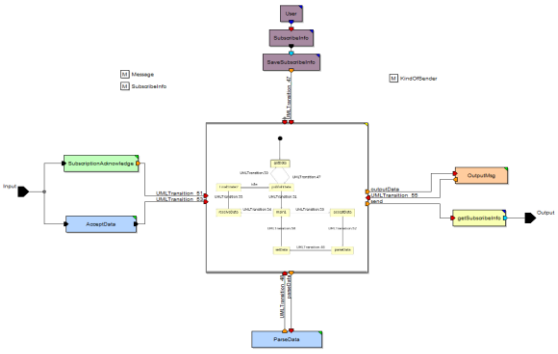

Figure 3: Constructed client model for proof of concept

The next step would be to add

- Bi directional capability
- Internal resource sizing, cache, memory, processor, bus, control, etc
- Build network detail based on ARMY input and description
- Architecture, type, throughput, latency, etc.
- Speed, bandwidth, performance, etc.
- Exercise with Traffic Generators
- Data, Video, Voice

Then we could run the model against full system to determine true performance against system boundaries to provide executable specifications

Requirements Based Engineering VS MBSE based

The summary below was taken from work at Airbus and Hamburg Aeronautics Institute comparing a pure requirement (UML) based system methodology vs. the use of MBSE and virtual prototypes using executable specs.

Requirements Based Engineering (RBE) is driven by the requirements and has separate written specifications at each design level. This leads to a misinterpretation by people at different design levels such as an integrator or a supplier. It does not show dependencies between requirements at each design level and dynamic behavior if not considered in the

early stages of design process. The system behavior is not considered under all conditions both normal and abnormal.

MBSE is driven by executable requirements and has an executable specification for system level down to each design level (abstraction to detail). This gives one an executable specification in a formal language which avoids misinterpretations, designs at lower design level directly show impacts on requirements at each design level, dynamic behavior can be investigated in early stages of design process and the operational environment is modeled by missions.

RBE design decisions are made based on experience and assumptions of engineers whereas MBSE design decisions are made by simulating different design alternatives and select best one (trade-off). The RBE functional breakdown and allocation to components is normally done by hand. MBSE allows for modeling and decisions on both the functional and architectural level. In RBE required system resources can't be identified adequately but in MBSE required system resources can be determined at architectural level. In RBE different models are used at different design levels but in MBSE the same model can be used at each design level. In RBE subsystems/components can't be validated within system context but in MBSE subsystems/components can be validated within system context at each level. One big issue in RBE is that documentation of non-functional requirements is not coherent unlike MBSE where non-functional requirements can be modeled and validated. In RBE test cases are derived from requirements and can be misinterpreted but in MBSE test cases can be derived from the unified model.

Examples of MBSE with VES

The following are military and industry examples of MBSE being used in the concept design phase as well as in root cause analysis to correct problems that could not be corrected in time and on budget using traditional methods.

PM Moving Tracking System (PM MTS)- Root Cause Analysis

PM MTS had an issue with the MTS software crashing the Network operating Center (NOC) when 650 units were in the Area of Interest (AOR). The vendor of the software contacted SAIC to see how SAIC could help. The issue was the cost to have 650+ MTS enabled vehicles for months at a time to do iterative tests and pay for drivers, vehicle costs and commercial satellite time during the testing period. This represented significant manpower, costs and time that PM MTS was not going to fund as this was a vendor software issue and PM MTS needed a fix that would be available in weeks not months. SAIC looked at all the data form the logs and built each message using MBSE techniques. Once these messages were complete a virtual vehicle was built that would send these messages in the volume and frequency as the AOR data that was provided. This virtual unit sent out GPS location updates and the message set that was created.

A 24 hour simulation was run with this data and the output compared to the AOR logs. They matched and thus validated the system. A lab was instantiated that contained a NOC, 5 MTS units and the MBSE computer. The transport used was 802.3 Ethernet so satellite latency would not be an issue. The NOC and 5 MTS units were tested to insure they behaved as expected.

The MBSE machine spawned a vehicle and it moved on the map and behaved like the real world MTS units. It sent point to point and group messages, it updated its location IAW the requirements of the software and the data from the AOR. The NOC and the 5 MTS units could not tell the difference from the spawned vehicle and the real MTS units. This Hardware-in-the-Loop (HITL) testing allowed the scaling of the network architecture beyond what is currently capable with live units in lab and field experimentation and observation of network behavior.

Once this was in place the MBSE spawned more units until the 650 unit threshold was reached and as expected the NOC crashed. The logs were examined and with the software vendor's developers and the SAIC MBSE team issues were found and corrected. This allowed iterative testing without the cost associated with live testing. Within two months' time the software was patched and attained a 2800 MTS Vehicle/AOR ratio. The lab was bought to PM MTS for a final demonstration and used three additional MTS enabled vans that drove around the post. The demo was accepted and the patch was made available and deployed in a week thus saving time, costs and manpower through iterative MBSE testing.

Robert Bosch/Siemens/Volkswagen-Audi Automotive Vetronics Platform Validation

With the introduction of advanced automotive telematics and vetronics to govern powertrain, entertainment, navigation, and control systems now necessitating over 100 electronic modules in the average vehicle, new techniques to prototype and simulation the behavior of such systems under various environmental situation are needed.

Robert Bosch, Infineon, Siemens Automotive Research, and Volkswagen/Audi funded an effort to create, through the use of MBSE, a virtual prototype to analyze new networked architectures on vehicles. These architectures looked at the coupling of CAN and FlexRay busses and the incorporation of switched Ethernet. Many system elements were modeled including the ABS, engine control unit, engine sensors, fuel consumption monitors, etc.

The most significant model consisted of the introduction of Automotive Open System Architecture (AUTOSAR) onto an existing platform. Similar to many impact studies of

adding new technologies onto existing platform architectures or new architectures accompanied by legacy systems, the ability to comprehend and analyze the dynamics such overhead impacts is crucial to validation of a design.

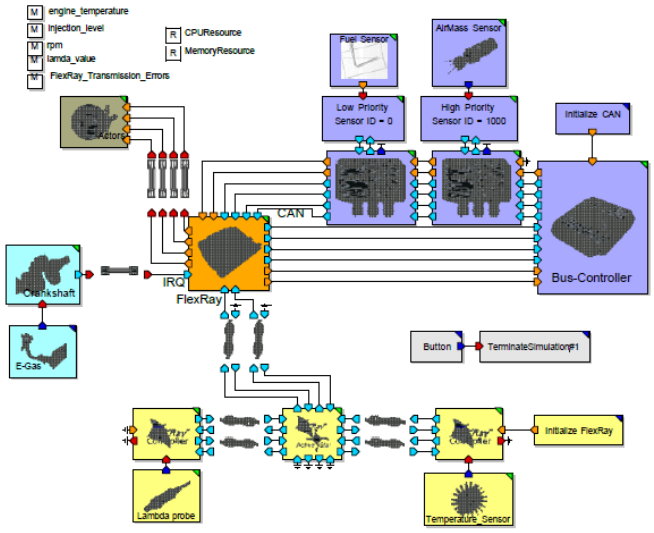

Figure 4 – Robert Bosch CAN Bus Model

Using MBSE a complete engine control system including all affected networked components and sensors were modeled. When simulated it was determined that the overhead of the newly employed AUTOSAR system taxed the CPU of the engine control system to the extent that it was dropping interrupt requests from the CAN bus. Since this happened only at certain speed thresholds, a complex coupled reaction would have been virtually impossible to catch using traditional engineering design disciplines until the hardware/software vehicle integration process. The conclusion supported the adoption of MBSE and the tool chain for automotive and vehicle systems.

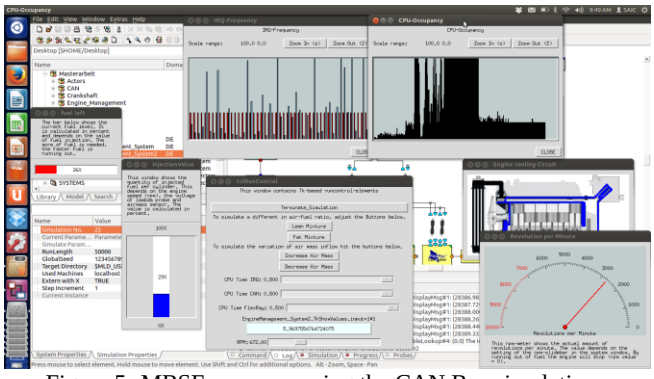

Figure 5- MBSE screen running the CAN Bus simulation

New technologies like FlexRay and AUTOSAR promises a lot when analyzed on their own. However, their value on the performance of the overall system and possible problems and costly product delays are generally only known when integrated in a system. In this project, an executable specification/virtual prototype of an engine control and management system is developed and used for the introduction of FlexRay and AUTOSAR. It was shown that FlexRay does not exhibit the increased delays of CAN at high RPM. However, the high overhead of AUTOSAR cannot be handled by the existing CPU. At less than half the maximum RMP the CPU is unable to cope with the increased processing load. When the library model of the CPU was replaced with a faster processor in the SoS model, the system performed nominally. The CPU upgrade allowed the incorporation of AUTOSAR into the new Audi A6/A8 vehicle platforms.

Airbus A350 Civilian Aircraft

The Airbus A350 consists of over 2500 networked electronic control units (ECU). These elements have to communicate within the aircraft structure as well as to ground systems for operations, maintenance, and health monitoring. The complexity, reliability, and response times needed to keep these systems operational required a new approach in determining which network architectures would provide the greatest systems availability. Traditional requirements based analysis and design choices were not able to accommodate the wide range of options and properly realize the interdependencies and coupled effects that each subsystem placed on the performance, power, and costing factors.

Airbus chose a MBSE methodology and toolset that supported the dynamic network coupling and through the adoption of VES were able to create an aircraft level model that functioned at the service level. The creation of a comprehensive SoS model that represented functionality and architecture provided through simulation the optimized placement, configuration, timing, performance, power, costing, and reliability of the aircraft architecture.

The backplane of the architecture utilized a fully modeled and validated Avionics Full-Duplex Switched Ethernet (AFDX) which offers switched, synchronous, secure Ethernet connections to vital control systems.

With an emphasis on an industry leading integrated communications capabilities for maintenance and operations, three candidate architectures were analyzed using MBSE. All were fully validated for cost, weight, power consumption, MCU/resource sizing, and heat dissipation.

The results proved the impact that a MBSE/VSE on the development schedule. The final selection for the A350 architecture optimized through MBSE showed a cost

reduction for avionics network of 70% through a reduction of cabling (68%), weight (30%), and MTBF/MTTR(10%).

The A350 made its maiden flight in May 2013 with no critical errors, it was developed on time and within budget proving the impact that adhering to executable sections, the creation of a comprehensive mission/service level SoS model afforded through MBSE can have on the success of one of the world's most complex commercial aircraft.

Performance item	Reference system	Optimized system
Cost of architecture [USD]	100%	28%
Reduction of cables [%]		68
Weight of architecture [kg]	280.06	201.00
Weight of cables [kg]	108.21	29.15
Length of cables [feet]	1,531.46	453.84
MTBF [h]	55,632.88	57534.15
MTTR [h]	31.63	29.89
MTBUR [h]	49,164.46	50,874.30
Availability	0,99	0,999999999999999

An optimized configuration will result in significant improvements in cost, weight, development time and risk level

Figure 6: SWaP-C savings using MBSE over traditional RBE

MBSE with VES as a VICTORY design enabler

Similar to AUTOSAR, the integration of various systems that employ the VICTORY architecture can benefit most from MBSE with VES. Because of the integration of so many systems that use common VICTORY services the sizing of the network becomes the challenge. Current methodologies of validation would take the basic GPS functions as services of position, orientation, direction of travel and time synch and multicast them via a VICTORY enabled GPS/DAGR device so that other consumers of this information can request and connect to those services and there is no discovery of any bottlenecks or interoperability issues until we actually do validation during integration and test. At that point the time and cost to correct these is far greater than the cost to validate during the concept and design phase as one could do with MBSE and VES. Conversely using MBSE in a bottom up development process (instead of using VES from concept development through life cycle management) cannot optimize a system at top level requirements. It also cannot avoid integration errors.

The VICTORY specification is an evolving specification with new releases coming approximately every 6 months. These releases add more functionality and have course corrections that are found while adding these new functionalities. For this reason it is important to have a functional baseline model so that interoperability and side effects (both intended and unintended) can be discovered and mitigated. In order to be able to keep up with new releases every 6 months and have the ability to test when the new VICTORY devices are not yet available MBSE with VES is essential for success.

For instance VICTORY is required to read CAN bus messages from the vehicle to obtain vehicle health. These messages are for basic engine functionality, oil pressure, coolant temp, tachometer, fuel level and the like. It is not efficient or expedient to test on a real vehicle so that the engine can run to send these messages out. There is not enough granularities to black box the system with a basic data generator. In order to properly size the power, the cooling and the amount of messages that will go into the VICTORY Data Bus (VDB) a more realistic real world environment must be used. MBSE with VES can be that environment. In the SAIC MBSE/VICTORY demo that was shown at GVSETs in August of 2014 the CAN Bus model described earlier was used to send real CAN messages into the SAIC VICTORY Management Software (VMS) system. The SAIC VMS centralizes the management of the VICTORY services on the VDB. The VMS contains the elements of the VDB to centralize the Web Service Description language (WSDL) and services on the bus. The SAIC VMS is compliant to the VICTORY 1.4.2 specification with a path to migrate to newer versions as they are released. The VMS is capable of dynamic changes to the system. This allows services to be modified (enabled/disabled) in real time. The VMS contains the Data, management and Health messages required to instantiate the VDB on the vehicle. It contains role based access control and VICTORY Compliant security measures. It contains the web server and WSDLs for the component type spec tags. The MBSE CAN Bus model feeds CAN messages and that model can be manipulated by moving the software sliders to regulate the engine speed, fuel mixture and air mass. The model also sends GPS 153 signals which are converted to VICTORY messages by the VMS. This allows the virtual vehicle to move on the VMS mapping program giving yaw, pitch, roll, rate of climb, direction and orientation.

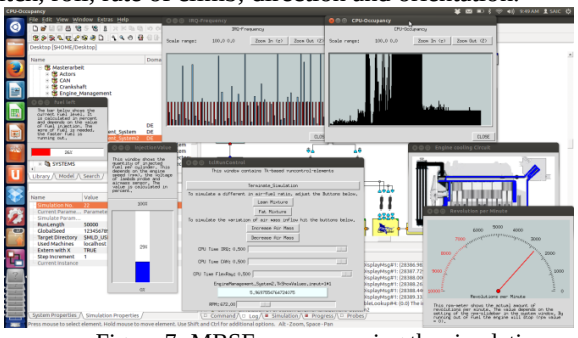

Figure 7- MBSE screen running the simulation

This Hardware in the Loop (HITL) bench allows the VMS to run VICTORY services in a lab without an engine or working GPS signal. As of the writing of this paper there are no VICTORY enabled GPS systems as Commercial Off the Shelf (COTS) devices so the integration labs would have to wait till one is available to include in this bench. Normally they would use a data generator but the labs would have to build one that sends VICTORY GPS messages and there would be no accurate model of resource availability in a SoS construct. By using a validated model of the CAN and GPS messages on a validated model of the Ethernet LAN (which includes the loss that occurs in Ethernet) you have the ability to model these messages and when the equipment is available they can be added to the HITL bench and removed from the model.

This agile methodology of build a little and test a little allows OEMs and integrators to work on a VICTORY enabled platform even though all the VICTORY enabled devices are not yet available. It also allows them to create baselines of the various versions of VICTORY to ensure their devices work with the new versions and are backwards compatible with the older versions. This is an implementation of Lab Based Risk Reduction (LBRR). New devices or newer specifications can be tested on the bench prior to integration to work out issues.

For exercises and events such as NIE this HITL VICTORY bench allows many different validated baseline configurations to be called up. This is the Golden Bench concept. One could recreate the scenarios that cause the field event to fail or put the hardware under test in the bench to see how it reacts to the Golden Bench configuration. This iterative, fast reconfiguration of the HITL bench using MBSE and VES will quickly find root cause analysis and allow “what-if” scenarios to be run.

Ultimately this can enable life cycle management that will provide demand forecasting and provisioning optimization for VICTORY and other programs based upon including fault and failure capabilities into the model and then running them against the already defined mission and scenarios that were devised at the concept stage. For VICTORY and other programs the Army could benefit from better inventory management both as a MBSE post implementation or integrated into the concept phase.

REFERENCES

[1] N. Fischer & H. Salzwedel, Validating Avionics Conceptual Architectures with Executable Specifications, Journal of Systems, Cybernetics and Informatics, Vol. 10, Number 4, 2012

[2] C. Grimm, Languages for System Specification: Selected Contributions on UML, SystemC, System Verilog, Mixed-Signal Systems, and Property Specifications from FDL'03, Springer Netherlands, 2004

[3] H. Salzwedel, R Sarkissian & G Fortney Interim Project Review PMO SBCT Modernization, Stryker Vehicle Electronics Feb 2013