Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

AUTONOMOUS AND COMPOSABLE M&S SYSTEM OF SYSTEMS WITH THE SIMULATION, EXPERIMENTATION, ANALYTICS AND TESTING (SEAT) FRAMEWORK

Saurabh Mittal
Nick Kasdaglis
Lamar Harrell
Robert L. Wittman
John Gibson
David Rocca

The MITRE Corporation
7515 Colshire Drive
McLean, VA 22102, USA

ABSTRACT

A simulation System of Systems (SoS) comprised of various simulation systems may have dissimilar modeling paradigms. For performing autonomy research and development, various types of simulation systems need to be brought together to build a multi-domain virtual SoS wherein effects of autonomous entities/agents can be observed. While the distributed simulation community has solved the integrability challenge using standards like Distributed Interactive Simulation (DIS) or High Level Architecture (HLA), the model composability challenge is an open research problem. Many software/systems can now be made available as docker applications which can be readily plugged into existing simulation systems. However, the trust issues with such integration limit their usage as established systems cannot trust third party apps. This paper highlights some of the challenges with building a cloud-based simulation SoS, proposes an architecture framework using the concept of structural autonomy and leverages Modeling & Simulation as a fundamental key enabler for autonomy research.

1 INTRODUCTION

Autonomy is the ability of a system to achieve goals while operating independently of external control (NASA 2015, David & Nielsen 2016); it is not Artificial Intelligence (AI), or automation, or adaptive behavior. A combination of various such capabilities to achieve the desired objective makes the autonomous system self-sufficient and self-directed. Designing a test-bed for testing autonomy is a challenging task, as is creating a virtual environment that realistically stimulates the system. Future complex system engineering will have to incorporate System of System (SoS) engineering methodologies that acknowledge that the constituent systems may have independent managerial, operational and evolutionary trajectories, are geographically displaced and produce emergent behaviors (Maier 2015). Modeling and Simulation is a core capability and a key enabler to explore SoS structure and behaviors (Mittal et al. 2008, Maier 2015, David and Nielsen 2016, and Mittal et al. 2017).

The Department of Defense (DoD) requires resources to implement and maintain such a networking infrastructure architecture for autonomous/AI systems development and innovation. From a Systems theory perspective, we consider autonomy at two fundamental levels: *behavioral autonomy* and *structural autonomy*. Behavioral autonomy can be understood through an autonomy spectrum with autonomy-at-rest on the far left and autonomy-in-motion on the far right (Grabowski 2015). Autonomy-at-rest implies AI-

based solutions executing portions of the observe-orient-decide-act (OODA) loop in the form of decision aides and cognitive assistants. The issue becomes more complicated when autonomy is defined with respect to a particular domain; for example, when developing an Army autonomous system, is the "autonomy" the same as in Navy autonomous systems? What happens when one tries to develop a multi-domain SoS with systems from multiple domains (e.g., ground, maritime, space, cyber and air)? Autonomy-in-motion further incorporates the mobility aspect on top of autonomy-at-rest capabilities. In addition to the kinematic mobility, the mobility may be relative to the environment; e.g., a cyber-virus "moves" through a network and has the ability to change the environment. Behavioral autonomy provides the ability to act in a tactical manner when an agent situates itself in the environment, either at rest or at motion, and develops relationships with other participating agents or systems. In a composed SoS, structural autonomy is the ability of the underlying infrastructure to autonomously incorporate autonomous software/system components in the SoS and to therefore provide behavioral autonomy for the larger SoS. This paper proposes an architecture framework and an execution test-bed to exploit the structural autonomy aspect of a SoS. The constituent systems may be autonomous systems preserving their autonomy and control, and need to be brought together in a collaborative development environment.

A foundational capability for future autonomous system development (Wierzabanowski 2020) is an open environment infrastructure for the integration, interoperability and composability of autonomous software/systems tools using an open systems architecture providing Live, Virtual and Constructive (LVC) execution. This environment will enable evolutionary and incremental improvements in the state of practice. It is necessary for autonomous systems to have a scalable and flexible foundation for integrating autonomous tools and capabilities within a multi-domain environment (i.e. an environment wherein systems from Army, Navy, Air Force etc. can be composed to deliver an extended SoS).

We present the **Simulation, Experimentation, Analytics and Testing (SEAT^{TM}) Framework** which we have created to provide a development and execution environment supporting the composability of interoperable simulation and analytical applications with the goal of performing useful simulation and analytical tasks for engineering autonomous systems. The simulation and analytical applications are anticipated to come from DoD and industry contributors bringing their skillsets to the SEAT environment in a container-based application form (i.e. Docker Container Platform) for flexibility of deployment and reduction of portability and dependency challenges. The SEAT environment has the following objectives:

- Provide a containerized distributed simulation platform to perform large scale on-demand experimentation for LVC scenarios.
- Realize syntactical, semantic and pragmatic interoperability in Joint All Domain Command and Control scenarios
- Build a plug'n'play architecture for external docker images (from vendors and collaborators) to be integrated on an as-needed basis and offer model and software interoperability through Docker containers, making simulation interoperability completely transparent by the use of distributed simulation standards like DIS/HLA.
- To encapsulate an abstract time simulation environment that can run both faster than real-time and in real-time.

We are not solving the model composability challenge at-large, but providing an infrastructure that does engineer simulation composability using existing distributed simulation standards and processes. Model composability is partially addressed through adoption of an agreed upon data model while simulation composability is addressed using simulation standards and protocols. We aim to address the challenges of: abstract-time execution, accessibility, automation for development, test and execution pipelines, heterogeneous tools, trust levels, architecture specification levels, multi-role and multi-user access and various processes for easy onboarding of participants to the shared collaborative SEAT environment.

2 HIGH LEVEL REQUIREMENTS AND VISION

The SEAT framework is guided by a three-pronged vision: the ability to bring together models and simulation systems at varying levels of specifications and structural autonomy, offer a simulation environment for external software/systems to play within the simulation, and be usable by various partners. Let us elaborate on each of the prongs.

2.1 Multi-Level Architectures and Abstraction Levels

In any complex systems modeling and simulation effort, models are built in an iterative and incremental manner. During this iterative cycle, knowledge, information and data is added to the model. Knowledge, information and data is analogous to the pragmatic, semantic and syntactic forms of linguistic interoperability (Gurr 1998, Mittal et al. 2008). If the model is not separated from the underlying simulator, this iteration is bounded by the simulation system that executes the model. However, once we separate the model from the simulator, we can explore the model's abstraction, resolution and fidelity levels required for the case-at-hand. Alternatively, selection of a simulation system that is not amenable for integration and interoperability with other simulators at a different resolution and fidelity can become a limiting factor in exploring a model's abstraction level. Ideally, we would like to achieve the progression (Figure 1) wherein we develop a reference model architecture with appropriate domain constraints to identify a broad set of options, parameters and concepts of operation (CONOPS) that need to be optimized and evaluated. At this first level, various domain-agnostic modeling and simulation methods and tools can be employed, such as formalism-based DEVS (Zeigler et al. 2000) tools or agent-based Netlogo (Wilensky and Rand 2015), to ensure the correct system behavior is observed. At the next level of Enterprise Architecture specification, we use the formal model from step 1 and add further requirements and constraints to architect an SoS with an objective of performance evaluation and domain simulator (e.g., OneSAF, JSAF, etc.) integration. At this level we bring different types of domain simulators to verify various domain-constraints, requirements specifications and their impact on the SoS performance. At the third and final level, we develop an SoS Solution Architecture that eventually is fielded. The SEAT framework must allow simulation at these architecture specification levels.

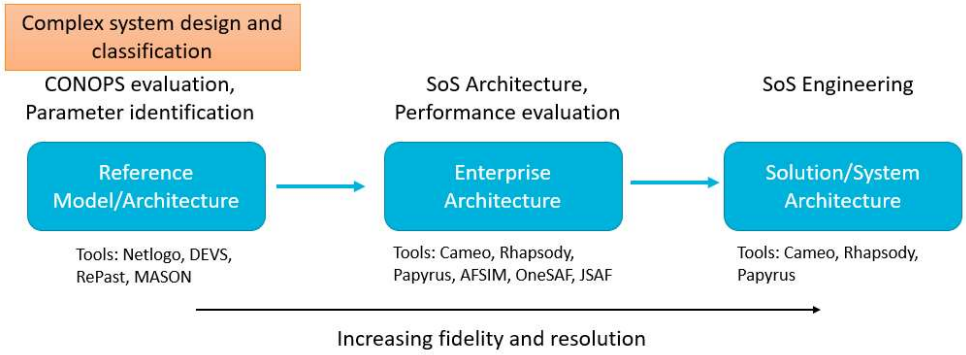

Figure 1. Architecture evaluation progression for multi-level abstractions.

2.2 M&S as a Core Capability

In a multi-player environment, each contributing system may provide different levels of autonomy. None of the applications or a single system in SoS can solve the whole problem. This is the fundamental aspect of any SoS. To preserve autonomy and be able to test (before deployment) with the larger SoS, a constituent system must be given a virtual SoS test-bed wherein integration tests could be performed (Step 2 in Figure 1). This warrants development of a simulation environment that provides the ground truth in a modeled scenario and interfaces with real-world data as well as with various domain specific and domain agnostic simulation systems for testing at various levels. Many SoS take the form of Complex Adaptive Systems (CAS) (Mittal and Martin 2017a) when the constituent systems are black-boxes and implement autonomous

functions across the entire spectrum (autonomy-at-rest to autonomy-in-motion). Figure 2 shows how a simulation ground truth is a key enabler for any system to test with if it wants to be a part of the larger SoS. The larger SoS (e.g., LVC) incorporates both the real-world data as well as the modeled data that is used to develop a hyperheuristic (as a black-box system) to experiment with both the developed model and the simulation model. The simulated scenario offers the ground truth for experimenting with the model both before and after the application of heuristic to evaluate the significant impact of the heuristic on the SoS. The SEAT framework must be able to offer ground truth for various apps and autonomous software/simulation systems to test out their capabilities in a black-box manner.

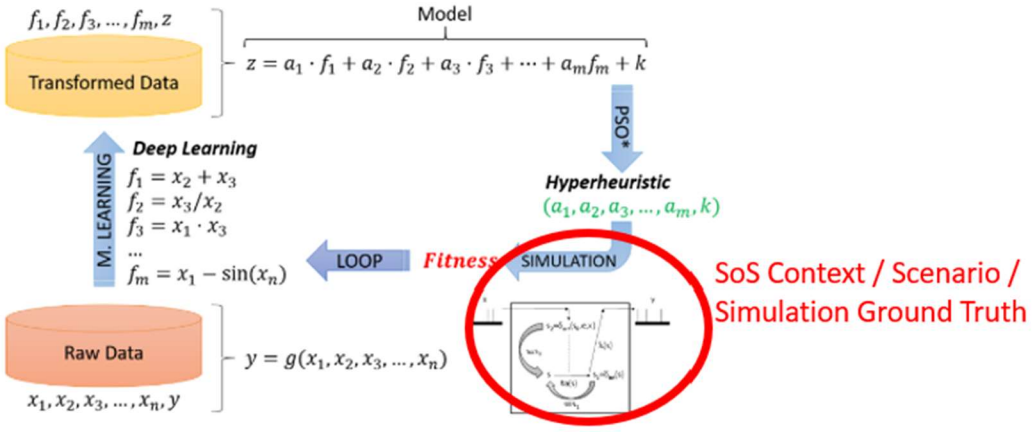

Figure 2: M&S as a core capability offering ground truth (adapted from Mittal and Martin 2017a).

2.3 Ease of Use

Any complex system will have many stakeholders. Likewise, in SEAT, there must be a provision for customized User Interfaces (UI) for different types of stakeholders. The three primary use case journey maps we need to consider are:

1. Researcher/warfighter: a domain-expert that may provide the model or the scenario and is interested in all aspects of the modeled SoS. He is also an evaluator and an end-user who wants to evaluate and test out a particular capability.
2. Developer: a software/systems engineer that builds the SEAT infrastructure and manages the release/build process through the Continuous Integration/Continuous Delivery (CI/CD) Development and Operations (DevOps) methodologies. This role is quite broad and may also involve simulation engineers and IT system engineer that install, test, evaluate and deploy simulation systems.
3. Industry: An external participant that contributes to the SEAT framework in either passive or active manner.

The SEAT framework must allow customized workflows for these three journey-maps, at a minimum, through a browser-based UI, as the infrastructure will be cloud-deployed. Figure 3 shows a user-oriented view of the SEAT framework. The SEAT implementation architecture shown in cloud deployed form is elaborated ahead in Section 6.

3 ARCHITECTURE FRAMEWORK

The SEAT execution environment is a container-based architecture wherein each container maintains its own autonomous execution environment, each supporting the structural autonomy concept. The container technology which has recently become the primary vehicle for leveraging cloud resources, offers many inherent advantages. For example, the associated DevOps processes, CI/CD pipelines, configuration

management, repository management, user management and remote access are some of the inherent capabilities. Additionally, the technical dependency management for each of the containers saves time when a software SoS is being architected. Each of the container applications comes bundled with its own set of dependencies and is shielded from other container apps.

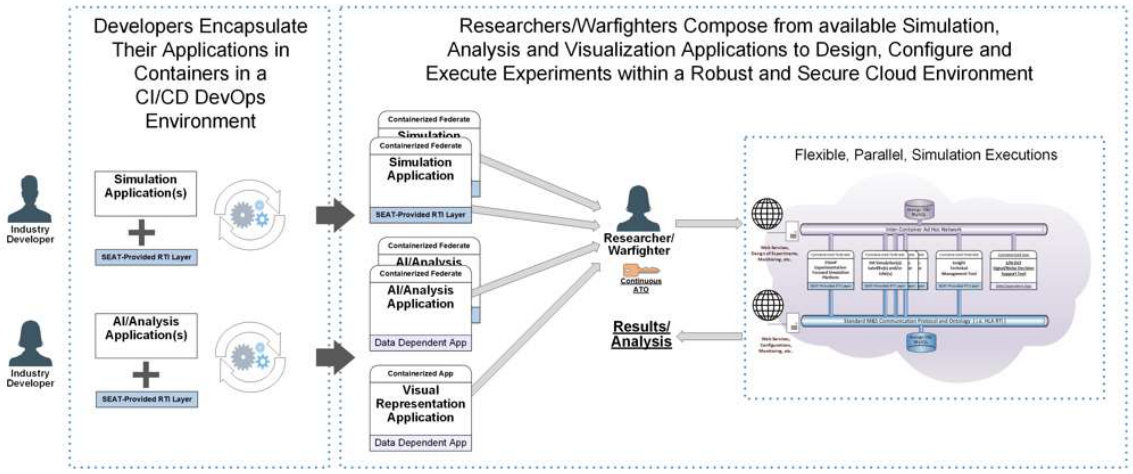

Figure 3: User oriented view of the SEAT Framework.

The SEAT architecture considers the container boundary vital and a critical requirement for software/systems that want to play within the SEAT. To "_grab a seat in SEAT_", one must have a container and an associated data model specification. As we address the software/systems integration and interoperability challenge, and have adopted the container technology, there are two modes of integration**:** **data dependent only** (*passive*/simulation-independent) and **data contributor** (*active*/simulation dependent). In data dependent only mode, the container app may only listen to the simulation ground truth. As a black-box, it can subscribe to the data channels and can use it to provide analytics or visualization capability. In the second data contributor mode, while the integration is dependent on the simulation data model, the interoperability issue is much bigger as the app data model may not conceptually align with the simulation data model. For example, consider bringing an Army simulation system trying to work with an Air Force simulation system, or as another example, a vehicle simulator with a cognitive simulation system for driver model. The conceptual alignment must happen before such simulation systems, or non-simulation system with similar conceptual dependencies are integrated. It is up to the system contributors to articulate, negotiate, and instantiate their necessary interfaces and processing mechanism to support these dependencies in the context of the SEAT infrastructure. This is necessary as the SEAT framework does address simulation composability using docker technology and available distributed simulation technology, but it does not address the model composition problem.

Figure 4 shows the layered architecture for SEAT framework. This layered M&S architecture is built using concepts described in Zeigler et al. (2000), Mittal et al. (2010) and Mittal and Martin (2017b). It categorically separates the modeling and simulation layers. It also adds the vertical User app and Security layers indicating the user apps can serve any of the horizontal layers and that the security aspect must be handled at both horizontal and vertical levels. The other layers are self-explanatory. The layers are also categorized into end-user and automation aspects, to emphasize the automation technologies that are available for simulation, cloud APIs and infrastructure levels.

The end-user layers are focused towards the user-roles as described in section 3. Many of the layers are marked dockerized to emphasize the use of container technology in those layers (Berg et al. 2017). The Visualization Services layer may or may not be dockerized as it may reside very close to the end-user. The visualization aspect is local to the end user as it may require considerable graphic processing power and it is impractical to mandate it as a docker container. The SEAT framework is offered as a services architecture in line with Modeling and Simulation as-a-Service (MSaaS) paradigm (Siegfried et al. 2018).

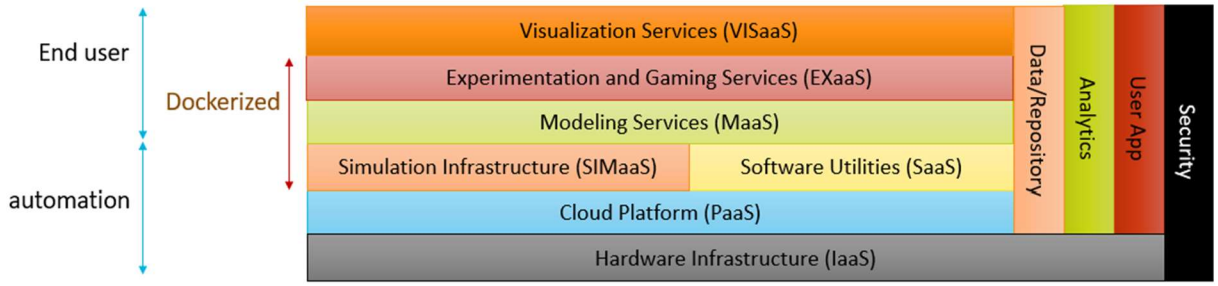

Figure 4: SEAT Layered Architecture Framework.

4 CONTINUOUS AUTHORITY TO OPERATE (CATO) PROCESS

We now outline the information assurance (IA) objectives, threats and countermeasures to secure the SEAT environment from external third-party applications developed for the SEAT environment. The objective is to have a SEAT environment where new or modified application can be rapidly approved and incorporated into the simulation environment through a process that achieves a Continuous Authority-to-Operate (Continuous ATO or CATO). There are two primary objectives for information assurance: *Rapid ATO* and *Application approval and security.* Rapid ATO should develop a process by which applications can receive an ATO rapidly. Application approval and security should provide a mechanism to integrate applications with minimal delay and without jeopardizing the security objectives.

4.1 Rapid ATO

Achieving a rapid ATO can be broken into two major areas: the CI/CD pipeline and the application software approval process. For purpose of an IA analysis, the SEAT framework can be viewed as a three-tiered system (Figure 5) composed of: (1) An operational SEAT environment that runs the simulations, (2) An infrastructure development CI/CD pipeline, and (3) An application development CI/CD pipeline.

The infrastructure and policies that require CI/CD pipeline approval are relatively static so we can significantly reduce the timeline to acquire the ATO as compared to the application software which changes much more frequently. Containerized servers instantiated by an approved CI/CD pipeline helps quickly develop new compliant infrastructure.

4.2 Application Approval, Development and Security

Application security begins with the software development life cycle (SDLC). SDLC encompasses the tools process, procedures and processes, (like the Building Security In Maturity Model (BSIMM 2019) is a potential way to evaluate the maturity of software development practices. Applications written without good software development processes may contain vulnerabilities that can be exploited by an attacker. *NIST SP800-53 Security and Privacy Controls for Federal Information Systems* (NIST 2013) *and SA-11 Developer Security Testing and Evaluation* (NIST 2019) require developers to create and follow plans for producing quality software. Third-party applications are developed by entities that may be external to the U.S Government or MITRE. To manage risk and develop trustworthiness, NIST provides guidance as an interdisciplinary risk model that incorporates the aspects of safety, reliability, resilience, cybersecurity and privacy (NIST 2017).

The SEAT infrastructure offers best practices to developers and restricts deployment within SEAT to those applications that follow best practices There are three primary security objectives: confidentiality (applications should not be able to be copied or examined by other applications), integrity (applications should not be modifiable by other applications) and availability (applications should not be able to deny approved connections or consume more than allotted resources). To provide application isolation for

achieving structural autonomy, streamlined ATO without impacting performance, a tiered trust model and an associated isolation mechanism needs to be developed. The level of trust placed in a new application depends on a number of factors: (1) the trustworthiness of the application development team and (2) how well that team adheres to good development practices. Other factors could be developed that affect the trust placed in a new application that incorporate data security procedures which protect the data exchanges with the application. Applications that are less trustworthy are isolated from the rest of the simulation environment and other applications and data. The level of trust drives the level of security isolation required. The lower the trust level, the more the application is isolated from the rest of the SEAT environment and other applications. Increasing levels of isolation will impact simulation performance and the allowed mechanisms for inter-application communication but allows for multi-level tradeoff between security and usability. Figure 6 shows technical architecture of the SEAT environment with respect to 3 trust levels within applications: fully trusted apps, medium untrusted apps, and untrusted apps. Fully trusted apps are available as white-boxes i.e. complete code and SDLC transparency. Medium trusted apps have limited code and SDLC transparency and may have dependencies that can never appear as white-boxes or validated. Untrusted apps may have a lot of dependences that haven't been validated and have questionable SDLC practices.

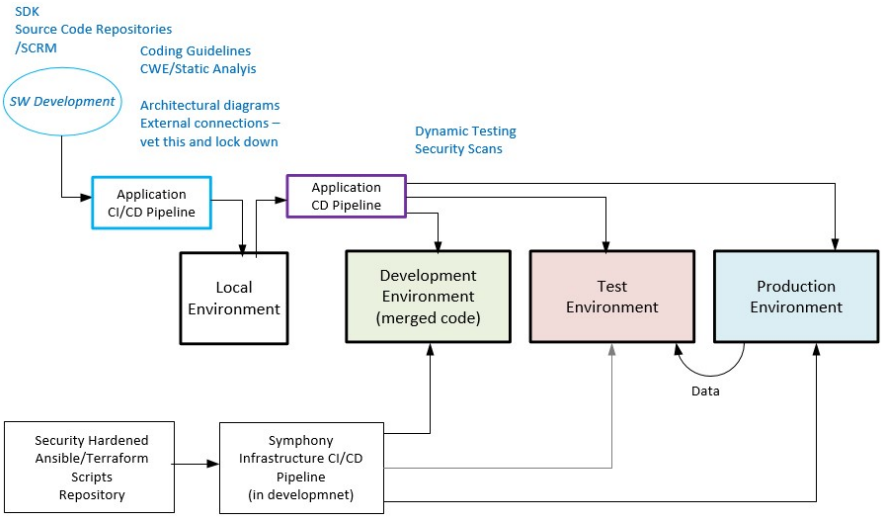

Figure 5: CI/CD pipelines and staging environments.

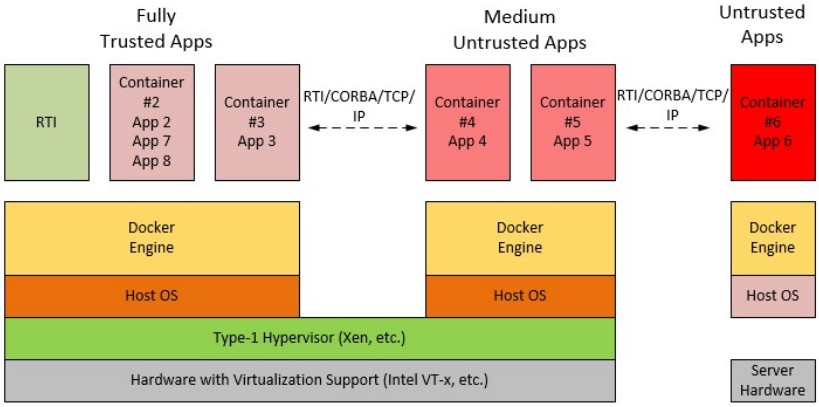

Figure 6. Isolation of Applications based on Trust levels.

Achieving a CATO for applications requires a shift towards working with dedicated assurors to become familiar with the system, the technology, the people and the risk tolerance of the Authorizing Official (AO). This creates a tight feedback loop between assurers and the DevOps team that allows for rapid assessment

and quick feedback to development teams. We are actively researching the mechanics and standards to make this process as efficient as possible and expect these concepts and architectural constraints to mature over time.

5 CLOUD INFRASTRUCTURE THROUGH MITRE SYMPHONY PLATFORM

MITRE developed an automated process called Symphony™ built on the CI/CD DevOps methodology (Symphony 2019). It provides Infrastructure as a Service (IaaS) capability to the SEAT architecture framework. The objective of CI/CD as powered by Symphony were two-fold: Create a Symphony enclave that can host custom docker images securely, and create a custom CI/CD pipeline for docker images that would confirm the security and integrity of docker images using industry leading tools before they enter the Symphony enclave.

Symphony is a preexisting MITRE framework that automates the provisioning of secure enclaves in the cloud. Symphony accelerates the process of gaining accreditation by providing pre-built, automated "recipes" to stand up the environment and software, as well as bundled documentation and security controls, in a matter of days. Some of the major security controls that Symphony provides out of the box are: centralized authentication, centralized logging of all machines and applications and centralized vulnerability scanning and virus scanning. Symphony can create any number of user virtual desktops that can be accessed easily through any modern web browser. The virtual desktops are similar to what would be found in a secure lab. They have no internet access, and are monitored and secured using the centralized authentication, logging and scanning. Symphony could be used to deploy enclaves to any Amazon Web Service (AWS) account, not just MITRE-owned ones, making it easy to create multiple SEAT environments to accommodate different communities based upon security or customer requirements. In addition Symphony is cross-provider and can deploy enclaves into Azure or on-premises VMware labs as well as AWS. This avoids cloud vendor lock-in and insulates SEAT from the differences in cloud providers.

For the SEAT framework, Symphony enables an accelerated initial ATO process and provides an infrastructure that can be used to implement CATO. All of these tools protect the integrity of the SEAT environment and allow for secure and controlled experiments to be run and monitored from the secure desktops. A custom Software Defined Networking (SDN) Zone was added to Symphony. This "SEAT Zone" functions as the network firewall separating the virtual machines that will be running the SEAT Experiments from the rest of the Symphony enclave.

5.1 Custom SEAT CI/CD Pipeline and CATO Support

A custom-built CI/CD pipeline was built to scan and test individual SEAT applications as they are being developed to ensure they will safely run inside of the Symphony enclave and not introduce vulnerabilities that could affect the implementation of CATO. The pipeline leverages the CI/CD tools provided by the Gitlab product family to perform the building and exporting of the custom SEAT Docker applications. Specifically, the Clair Docker Image Scanner was implemented and integrated into the pipeline and was used to perform a full known vulnerability scan of the custom images as they are being developed. Figure 7 shows the implemented cloud architecture.

The MITRE Symphony platform includes the automation, security (i.e., ATO or CATO) support and flexibility of cloud deployment (i.e., AWS) useful to the long-term goals of the SEAT environment. The Symphony platform provided a development and deployment environment surpassing the originally conceived architecture in several practical ways. In Figure 7, the "Gitlab CI/CD Pipeline" includes Git utilities and an automated build, test and report processing workflow toward generation of container-based simulation and analysis assets (i.e., input data and Docker images). Once set up by a SEAT Application Developer with a unique configuration file along with necessary Docker files, the system will automatically propagate application changes through to an AWS "S3 bucket" to await uploading to the "Symphony Enclave" to enable composable, interoperable deployment for simulation experimentation. Symphony specified an accelerated initial ATO and provided an infrastructure that can implement CATO for the CI/CD Pipeline. This is largely possible because the CI/CD pipeline is relatively isolated and accessible only to

authorized users. This is beneficial for security, but may not be as useful for software development. Subsequently, the simulation and analysis assets (i.e., data and Docker containers) will then be able to be composed into experiments executing in the cloud-based Symphony Enclave designed by a researcher to help evaluate choices for some operational context.

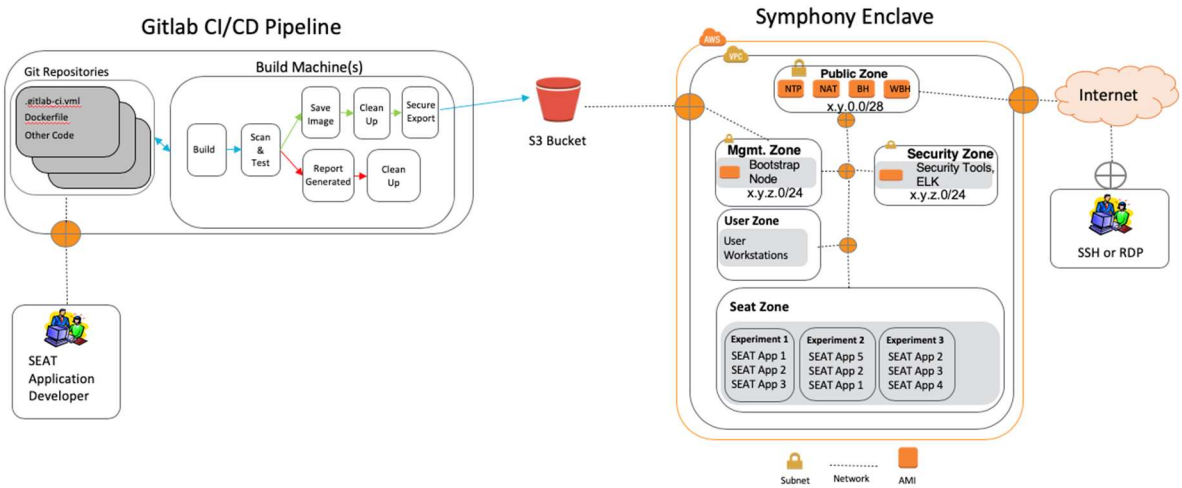

Figure 7: SEAT framework cloud architecture.

6 IMPLEMENTATION

We will describe a sample application within the SEAT execution environment in a SEAT project-enclave, labeled as Symphony Enclave in Figure 7. A SEAT project-enclave is an instantiation of SEAT environment for a specific project. It is an on-demand deployment of containerized M&S infrastructure in a virtualized environment.

6.1 Experiment Simulation Platform (ESimP)

The Experiment Simulation Platform (ESimP) was developed to evaluate the effect of small unmanned ground and airborne vehicles (UxVs) in a ground tactical combat scenario (Figure 8). The UxVs would be evaluated in a jamming environment as well as a permissive communications environment, and that the UxVs would utilize different levels of autonomy. A goal of ESimP is that it be used for flexible, quick prototyping of these varying combat scenarios (i.e., at a reference model architecture level). The platform's ability to track the actions of individual entities allows us to evaluate the precise events that led to a certain action (e.g., kill chains), is a key advantage over similar models. ESimP was written in NetLogo, a multi-agent programmable modeling environment, developed at Northwestern University. Combat entities and unmanned vehicles are represented as heterogenous agents with their own set of rules/behaviors. In this summary, we refer to both combat entities and unmanned vehicles as "entities," and communication between entities is represented as links. Links and entities can be grouped into a variety of networks (command and control, situational awareness or simply communications), and in the simulation, the networks are used to determine communication paths, the data available within those paths, the vulnerability of those paths as well as other networking attributes between entities. With this explicit--albeit abstract representation of communications networks--we are able to use ESimP to investigate aspects of electronic measures and countermeasures and their impact on information flow, and ultimately on the represented entity and unit's operational mission effectiveness. ESimP currently provides a platform for prototyping various combat scenarios that exercise, track, and show impact of both kinetic and non-kinetic aspects of mission operations. ESimP has also been used as a test-harness within the development of a spectrum dependency map (SDM) decision support tool prototype, allowing the initialization of simulated radio band assignments, and their operational status from SDM decision support tool data.

6.2 Bringing ESimP into the SEAT Environment

Much of the integration activity involved understanding the requirements, dependencies, expected simulation output and analytic applications and containerizing the simulation systems architecture through CI/CD pipelines for seamless deployment. For example, the NetLogo simulation framework application with multiple parallel execution experimental runs points to a container-based "headless" deployment (without the User Interface) of NetLogo ESimP within the SEAT environment. For the *passive* mode integration in SEAT, a database approach (i.e. mongo DB or MySQL) allows applications to access and operate on simulation execution results for additional analytic or simulation tasks. For the *active* data contributor model, the SEAT environment makes use of an HLA Federate Object Model (FOM) based on the Real-time Platform Representation (RPR) data model as a robust and tested manner of providing the ontology and syntax for real-time interaction between the modeled entities. The Cloud inset in Figure 3 shows the technical simulation architecture for both passive and active modes.

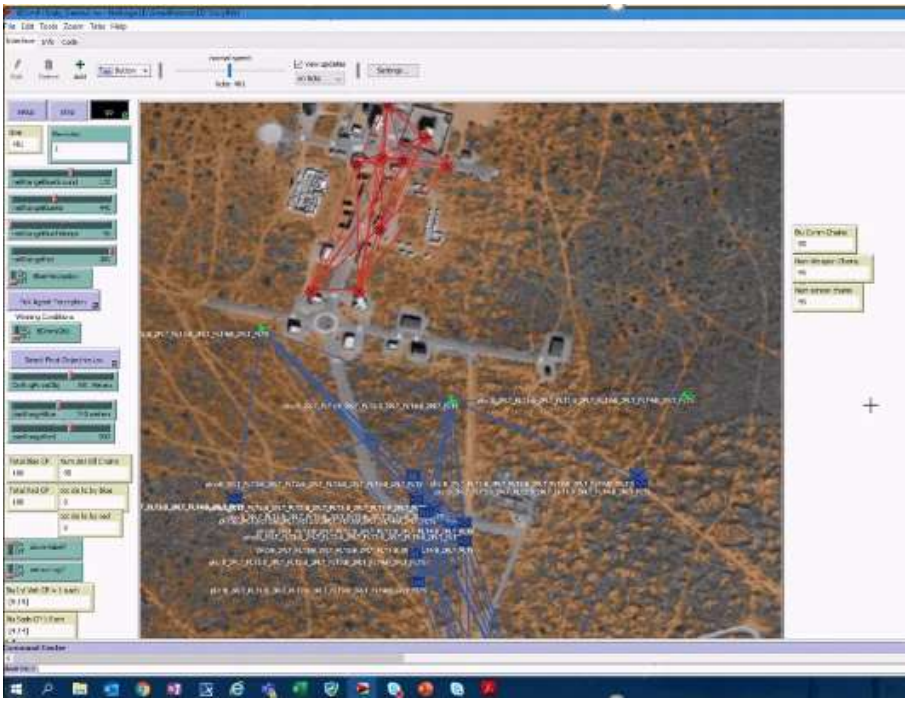

Figure 8: Snapshot showing the ESimP modeled application.

The newly provisioned SEAT-Symphony platform has been exercised successfully with an example container-based simulation application within the SEAT platform effort. Once the SEAT-Symphony unique configuration file is edited, and with a successful connection to the development Git repository, the CI/CD Pipeline is set to automatically perform a series of processing stages with each stage giving the SEAT developer feedback on success or failure. The Java-NetLogo/ESimP/HLA Docker image was then executed through the Vulnerability assessment tool to incorporate the CATO process. The identified vulnerabilities were marked and the image was validated. It was then uploaded from the S3 bucket to the Symphony enclave by the Symphony enclave manager for experiment execution. The ESimP simulation ran successfully to completion and output expected results. The complete pipeline was executed: from initial docker image building, to running securely on the Symphony enclave with the ESimP headless custom docker image. For this initial cycle, the entire ESimP application was a single federate. This is important as it demonstrates that the SEAT environment can build any custom images, execute the established CATO process for both infrastructure and application CI/CD pipelines and make it available for further integration with other external docker images.

ESimP/HLA bridge was developed to interface the Netlogo simulator with HLA infrastructure. Containerizing HLA infrastructure has been reported by Berg et al. (2017). In the next iteration ESimP/HLA will interact with other HLA federates and will be reported in our follow-on work.

7 CONCLUSION

Autonomy research and development requires a testbed that can perform SoS engineering at different abstraction, resolution and fidelity levels. Behavioral autonomy is being studied in disciplines like Cognitive Science and Artificial Intelligence. Structural autonomy at the infrastructure level must be supported by a test-bed that mimics autonomous behaviors at multiple levels of system specifications. M&S is a key enabler that provides the techniques, technologies and procedures to develop a virtual testbed in which various autonomous software/systems can be tested, evaluated, verified and validated. The entire software and systems engineering community is making strides to migrate to cloud-based systems and M&S is no exception. The proposed SEAT framework aims to provide the virtual testbed wherein one can perform simulation, experimentation, analytics and testing for different architecture model specifications. The framework exploits the ground truth M&S capability and offers role-based access to different stakeholders, and test interoperability and composability of the new autonomous systems integrating with the larger SoS.

This paper described the SEAT layered architecture framework that employs docker technology, provides description of the Cloud infrastructure through MITRE's Symphony platform, conceptualizes various trust levels wherein external third party apps can be CATO'd in the SEAT environment and become a part of the composed SoS. The concepts and architecture were substantiated by bringing an existing simulation application, ESimP, into the SEAT execution environment and the integration process we underwent to perform the onboarding.

While we demonstrated how one simulation in single domain application can be onboarded and provide the ground truth for another docker app, we still need to describe a multi-domain and multi-abstraction case study wherein we bring high fidelity simulators to do SoS M&S-based performance engineering between two different domains, e.g. Army and Air Force. Performance evaluation needs to be done to investigate the applicability in LVC systems. Finally, trust, composability and interoperability have to be addressed in more detail to achieve the vision.

ACKNOWLEDGMENTS

The development of the SEAT project was funded under the Office of the Undersecretary of Defense, Research and Engineering 'Autonomy and C4I Community of Interest (CoI)'. We wish to acknowledge Jean-Charles (JC) Ledé, AFRL Autonomy Technical Advisor, Autonomy Community of Interest Lead and Dr. Mark Linderman, C4I Community of Interest Lead. We would also like to thank various team members that supported the development of this work: Nichole Davis, Brian Wickham, Tobin Bergen-Hill, Sim Dy, Jonathan Byington, Emmet Beeker, Kevin Bergollo, Alex Ethier, Susan Kuwana and Jon Bond.

This technical data was produced for the U. S. Government under Contract No. FA8702-19-C-0001, and is subject to the Rights in Technical Data-Noncommercial Items Clause DFARS 252.227-7013 (FEB 2014). The views, opinions, and/or findings contained in this paper are those of The MITRE Corporation and should not be construed as an official government position, policy, or decision, unless designated by other documentation.

REFERENCES

Berg, T., Siegel, B., Cramp, A., (2017) Containerization of High Level Architecture-based Simulations: A Case Study, *Journal of Defense Modeling and Simulation*, Vol.14, Issue 2, pp. 115-138.

BSIMM (2019) Building Security Maturity Model, www.bsimm.com

David, R., Nielsen, P. (2016) Summer Study on Autonomy, Defense Science Board, Office of Under Secretary of Defense, USA

Grabowski, R., (2015) Big Picture for Autonomy Research in DoD, Soft and Secure Systems and Software Symposium

Maier, M. (2015) The Role of Modeling and Simulation in System of Systems Development, in L. Rainey, A. Tolk (eds.) *Modeling and Simulation Support for System of Systems Engineering*, Wiley & Sons, NJ.

Mittal, S., Zeigler, B.P., Martin, J.L.R., Sahin, F., Jamshidi, M., (2008) Modeling and Simulation for System of Systems Engineering, in M. Jamshidi (ed.) *System of Systems Engineering for 21st Century*, Wiley & Sons, NJ.

Mittal, S., Zeigler, B.P., Martin, J.L.R. (2010) Implementation of Formal Standard for Interoperability in M&S/system of systems integration with DEVS/SOA, *International Command and Control Journal*, 3(1)

Mittal, S., Diallo, S., Tolk. A. (2017) *Emergent Behavior in Complex Systems Engineering: A Modeling and Simulation Approach*, Wiley & Sons, NJ.

Mittal, S., Martin, J.L.R (2017a) Simulation-based Complex Adaptive Systems, in S. Mittal, U. Durak, T. Oren (eds.) *Guide to Simulation-based Disciplines: Advancing our Computational Future*, Springer.

Mittal, S., Martin, J.L.R. (2017b) DEVSML 3.0: Rapid Deployment of DEVS Farm in Cloud Environment using Microservices and Containers, Spring Simulation Multi-conference, Virginia Beach, VA

NASA (2015) NASA Technology Roadmaps

NIST (2017) NIST SP1500-202 Framework for Cyber-Physical Systems: Volume 2, Working Group Reports Ver. 1.0

NIST (2013) NIST SP800-53 Security and Privacy Controls for Federal Information Systems

NIST (2019) NIST Special Publication 800-53 (Rev. 4) SA-11 Developer Security Testing and Evaluation

Siegfried, R., Lloyd, J., Berg, T. V. (2018) A New Reality: Modelling and Simulation as a Service, *Journal of Cyber Security and Information Systems*, 6 (3).

Symphony (2019) https://www.mitre.org/capabilities/advanced-technologies/symphony-automated-secure-platform

Wierzbanowski, S., (2020) Collaborative Operations in Denied Environment (CODE) at https://www.darpa.mil/program/collaborative-operations-in-denied-environment, (last accessed January 31, 2020)

Wilensky, U., Rand, B., (2015) *Introduction to Agent-based Modeling: Modeling Natural, Social and Engineering Complex Systems with Netlogo*, MIT Press, MA.

Zeigler, B. P., Praehofer, H., Kim, T., (2000) *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press.

AUTHOR BIOGRAPHIES

SAURABH MITTAL is the Chief Scientist for Simulation, Experimentation and Gaming Department at the MITRE Corporation. He holds both Ph.D. and M.S in Electrical and Computer Engineering from the University of Arizona, Tucson. His email address is smittal@mitre.org.

NICK KASDAGLIS is a Principal Cognitive Engineer and Group Leader in Simulation, Experimentation and Gaming Department at the MITRE Corporation. He holds a Ph.D. in Human Centered Design and M.S. in Aviation Human Factors from Florida Institute of Technology. His email address is nkasdaglis@mitre.org.

LAMAR HARELL is a Lead Simulation Integration Engineer for the MITRE Corporation. He holds an M.S. In Industrial Engineering from the University of Central Florida, and a B.S. in Industrial Design from the Georgia Institute of Technology. His email address is wharrell@mitre.org.

ROBERT L. WITTMAN is the Head of the Simulation, Experimentation and Gaming Department for the MITRE Corporation. He holds a Ph.D. in Industrial Engineering from the University of Central Florida, an M.S. in Software Engineering from the University of West Florida, and a B.S in Computer Science from Washington State University. His email address is rwittman@mitre.org.

JOHN GIBSON is a Lead Software Engineer at the MITRE Corporation. He holds a BA in Computer Science from UC Berkeley. His research interests include security and DevOps. His email address is jgibson@mitre.org.

DAVID ROCCA is a Sr. Software Engineer for the MITRE Corporation. He is pursuing his M.S. in Computer Science at University of Massachusetts Lowell (UML) and holds a B.S in Computer Science also from UML. His email address is drocca@mitre.org.