Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

RUNTIME EXECUTION MANAGEMENT OF DISTRIBUTED SIMULATIONS

Keith Snively

Dynamic Animation Systems, Inc.
Fairfax, VA, USA

Richard Leslie

Kinex Inc.
Manassas, VA, USA

Chris Gaughan

Army Research Laboratory
Human Research and Engineering Directorate
Simulation and Training Technology Center
Orlando, FL, USA

ABSTRACT

Distributed Modeling and Simulation (M&S) provides benefit from the ability to bring together a large number of simulations, across a network, to fulfill a specific requirement. However, this capability comes with the costs and complexity of coordinating all of the computing platforms for the startup, execution, shutdown and artifact collection of the simulation execution. Typically, an exercise event also requires many iterations of the simulation execution, necessitating the ability to perform these tasks in an efficient and repeatable manner. This paper discusses an approach to handle the runtime execution of a simulation exercise as part of the Executable Architecture Systems Engineering (EASE) research project. We discuss the methodologies used to control the overall execution of a distributed simulation as well as control the individual applications involved. We further present some of the current use cases for this approach and lessons identified.

1 INTRODUCTION

Distributed Modeling and Simulation (M&S) supports training, systems analysis and concept exploration, to name a few applications. Towards this end, composing large, distributed simulation events can be a complex task. The simulation engineer must design or choose a scenario that meets the desired functional capabilities and then determine the simulations that meet those criteria. Assets must be acquired to support each simulation application and process. The assets may be computers located in a simulation lab, virtual machines or a combination of both. Proper network connectivity and capability must also be ensured. The simulation engineer must also determine startup, execution and shutdown procedures as well as how artifacts will be collected for each iteration.

To lower the human and material costs of leveraging M&S, the U.S. Army Research Laboratory (ARL) Human Research and Engineering Directorate (HRED) Simulation and Training Technology Center (STTC) is conducting the Executable Architecture Systems Engineering (EASE) research project. The goal of the project is to provide an executable architecture based on systems engineering for M&S. The approach allows users to traverse systems engineering information to compose and execute simulation scenarios that address their analysis or training goals (Gallant, Metevier and Gaughan 2011).

The EASE architecture consists of a number of components. The Software Design Description (SDD) captures the systems engineering information on the available simulation applications, their capa-

bilities and how they interoperate in a simulation environment. The SDD also allows the system engineering user to add new simulation applications (Beauchat et al. 2012).

The EASE Interview System allows the user to traverse captured system engineering information to select and compose a simulation system. The user is presented with a list of options based upon scenario criteria and functional capabilities and has the ability to customize components of the scenario (Gallogly et. al. 2012). Additional advanced capabilities allow the user to inject custom properties and create surrogates to fill in specific capabilities.

Once the scenario has been designed and the components chosen, The EASE Deployment Management System determines the necessary assets for execution and deploys software and configuration files. It employs Platform as a Service (PaaS) to utilize virtual and hardware assets in support of a simulation exercise (Murphy, Diego and Gallant 2011). Its tasking service then determines how and when to run a simulation execution.

Finally, the EASE Coordinator is responsible for the actual execution of the simulation exercise. The Coordinator handles the Time Sequence of Events provided by the tasking system and controls the launch, initialization, shutdown and cleanup of each process. The Coordinator is also responsible for progression of the overall simulation execution ensuring all processes perform the necessary tasks at the proper time. This paper will discuss the EASE Coordinator in detail.

2 COORDINATOR

2.1 Problem Space

The runtime execution of a large distributed simulation can be a complex task. Orderly management of the startup, execution, shutdown and data collection tasks are required to support iterations of a simulation run. In addition, unanticipated error conditions must be monitored to determine when an iteration should be terminated and restarted. Often these tasks are handled manually. An example is the US Army Training and Doctrine Command (TRADOC) Maneuver Support Center of Excellence (MSCoE) in supporting the Battle Laboratory Collaborative Simulation Environment (BLCSE) experiments. In the current exercise, MSCoE will run three clusters of OneSAF simulators, where each cluster consists of a Battlemaster station, several backend simulation cores and an interoperability component that connects to a High Level Architecture (HLA) Run-Time Infrastructure (RTI) for communicating with other laboratories (IEEE 2000). This configuration requires support personnel running processes on roughly two dozen host machines. In addition, scenarios must be loaded and initialized on the three Battlemaster hosts. Once complete, all the application processes must be shut down and any data artifacts collected. The support staff must also monitor the simulations to make sure applications are restarted if a fault occurs. In order to support this exercise, MSCoE is currently bringing in extra staff. In addition, the process can be subject to human error in starting the required processes in the proper order with the required configuration.

In other cases, the application management is more automated. The initial version of the EASE Deploy Management System used a sequence of events based the Quartz scheduler (Terracotta 2013). This works well for process execution and sequencing them in time. Unfortunately, it lacks an ability to react to external events, such as a user deciding to shut down the execution or the simulation reaching a given objective. It is also harder to handle application monitoring and fault recovery where some startup steps may need to be repeated.

2.2 Applying State Charts

To improve the automation for the management of a simulation exercise, the Coordinator utilizes Harel State Charts to describe the execution of a simulation and its component applications. Harel State Tables allow for hierarchically nested states and include associated activities for states and transitions (Harel 1987). They also form the basis of the Unified Modeling Language (UML) State-Diagrams (OMG 2009).

State charts readily lend themselves to description of distributed simulations and its components and are used within HLA specifications. The Coordinator uses a State Chart XML (SCXML) to represent each state chart. SCXML specifies a generic state-machine execution based on Harel State Tables expressed in XML (W3C 2012). SCXML is an evolving standard supported by The World Wide Web Consortium (W3C). There are numerous software products which provide the execution of SCXML documents supporting a variety of programming languages. Currently, the Coordinator uses the Apache Commons SCXML library as the state machine engine. The library is written in Java and provides numerous extension points, including the ability to add custom actions, event transports and semantics.

Within the Coordinator, a series of state charts are used to represent the simulation execution and each process participating in the execution. Each state chart, represented as SCXML, executes asynchronously within an SCXML engine. Custom extensions to the Apache Commons SCXML library allow for the spawning of other SCXML engines and exchanging of external events with these other SCXML engines. The events may be sent between engines within the same process or across a network. The events can be used as triggers to transition to new states, set internal data or convey error conditions. Figure 1 depicts the execution of the state machines within the Coordinator architecture.

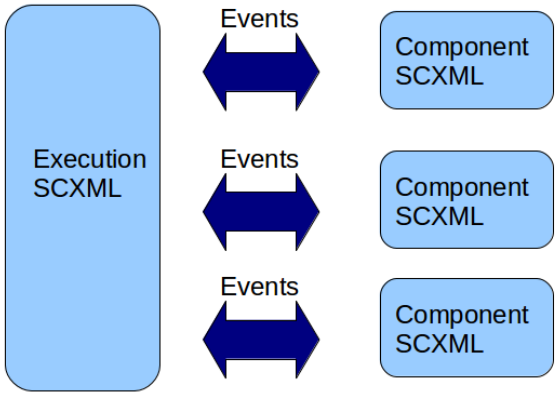

Figure 1: Coordinator SCXML engines

As depicted, the overall execution of the simulation scenario is represented as an SCXML document. The execution SCXML specifies the available states for the execution and the criteria for transitioning between the states. The execution SCXML engine sends events to and receives events from the components SCXML engines. These event exchanges progress the execution and components through their respective states. Error events may also be sent and handled as appropriate for the execution and error condition encountered. Figure 2 shows a high level sample execution where high level states are provided based upon the requirements of the previous section.

A number of the states are composite, namely *Startup*, *Execute*, *Shutdown*, *CollectData* and *Halt*. Each of these composite states invokes a sub-SCXML engine which executes the behavior for that state. These sub-SCXML engines run to completion before the state under which they are invoked transitions, much like a composite state. The customizable behavior given in the Coordinator XML for the simulation execution lies within these sub-SCXML engines. Using a sub-SCXML specification simplifies the respective files.

In addition, each process participating in an execution, referred to as a component, is represented by an SCXML document. There are three types of components in the Coordinator: application, environment and process.

Application components designate simulation applications, such as the OneSAF Force-on-Force simulation (Wittman and Harrison 2001), that directly participate in an execution. Generally, the application participates through the full course of a simulation execution run.

Environment components refer to components that are required to create the simulation execution infrastructure. An example of an environment component would be a centralized process to support the communication protocol, such as HLA or TENA (Powell and Noseworthy 2012). Typically these would be the first set of processes launched for an execution run and the last to be shutdown.

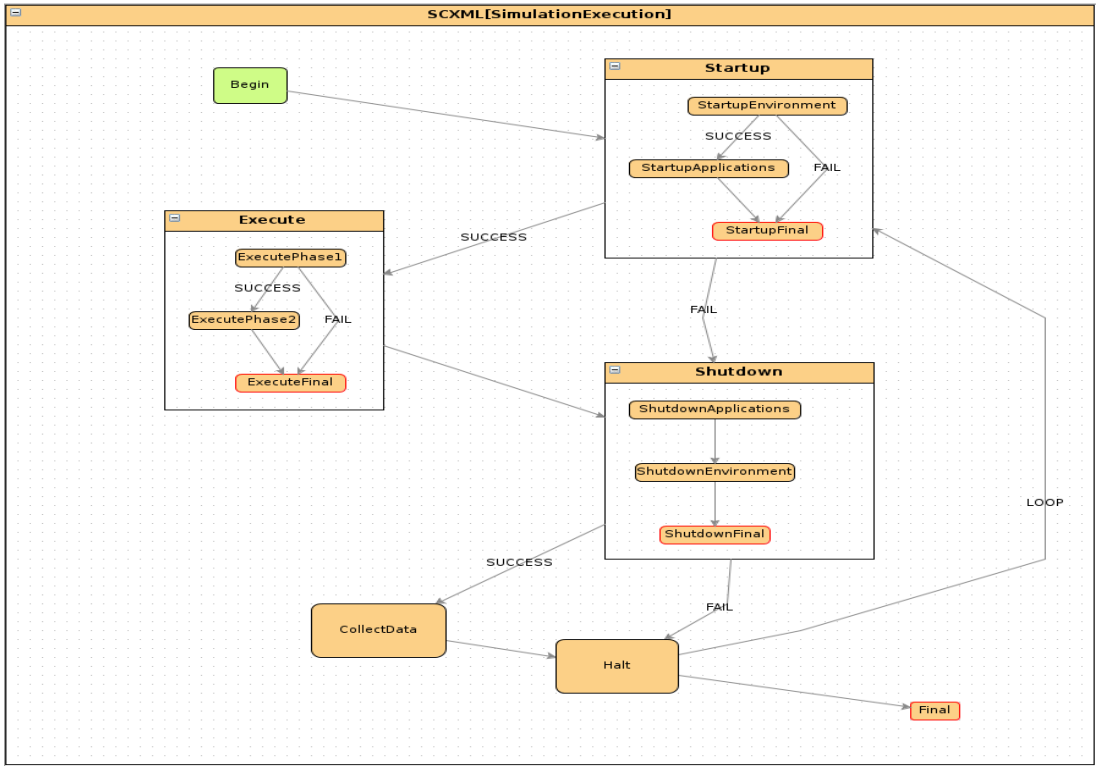

Figure 2: Simulation execution SCXML

Process components are generic processes that run during a simulation execution. These can be transient or long term processes. An example would be the execution of a script that transforms data collected during an execution run. Another example would be executing an application that sends commands to the simulation execution and then exits, such as an Advanced Testing Capability (ATC) test case (Hurt et al. 2006). Figure 3 shows an example of the application SCXML.

The Initialize, LoadScenario, Execute, Shutdown and Halt states are compound. The actions performed in these states may be customized behavior in the Coordinator XML.

2.3 SCXML Generation

The State Chart representations shown in the previous section, while powerful, are also complex. Therefore, these representation are templatized and generated from a simplified XML input designated as the Coordinator. This Coordinator XML file, supplied by the user or invoking process, specifies the components and their behavior in the execution and the behavior of the execution itself. The Coordinator uses XSL Transform (XSLT) 2.0 to generate the required SCXML documents for an execution from the Coordinator XML (W3C 2007). Additional configuration information is also generated from the input as re-

quired by the Coordinator for executing the simulation. Finally, the input may specify addresses to post status and results of execution.

Within the Coordinator XML, a set of actions can be specified for each available high level state of the component. Similarly, for the execution, a set of actions to be performed can be specified for each high level execution state. The high level states available for components and the execution are determined by the template SCXML documents and are based upon the role. Below is an excerpt for an application specification.

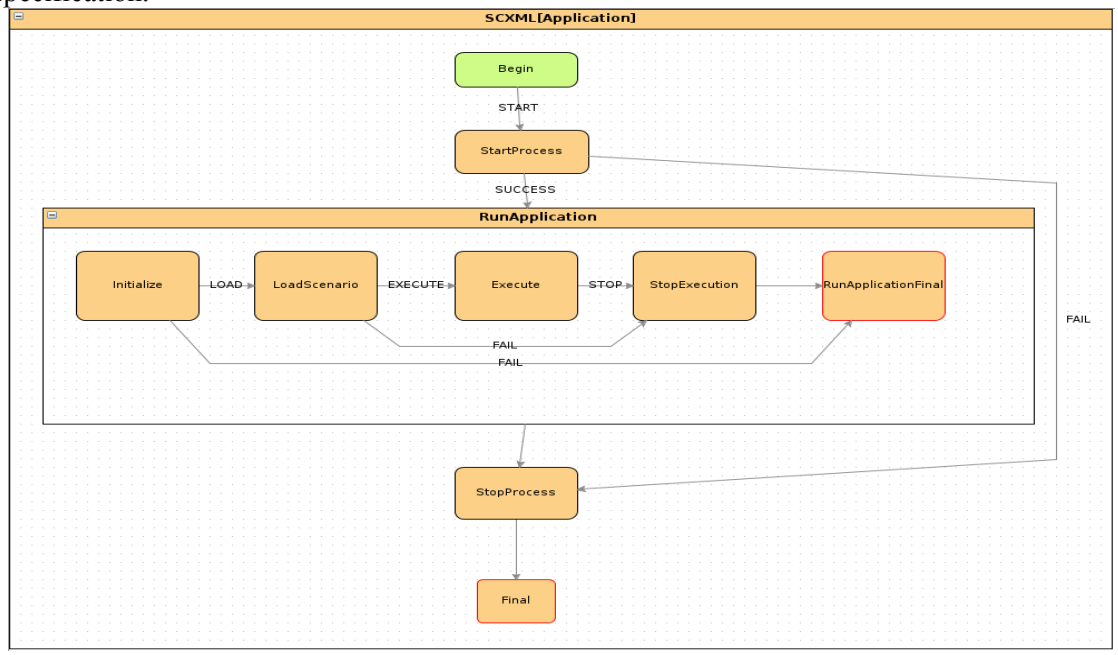

Figure 3: Application SCXML

```
<!--
        application: Specify behavior of a application component, e.g. a
        federate.
        @id:  A unique identifier for this component
-->
 <application id="sampleApp_1">
      <launch>
          <sshStart host=”machineA”
                    dir="/usr/local/ease/coordinator/examples/scripts"
                    script="sampleAppStart.sh" />
      </launch>
      <initialize>
      </initialize>
      <execute>
      </execute>
      <shutdown>
          <sshStop dir="/usr/local/ease/coordinator/examples/scripts"
                   script="sampleAppStop.sh"/>
      </shutdown>
      <halt>
          <sshStop dir="/usr/local/ease/coordinator/examples/scripts"
                   script=" sampleAppStop.sh "/>
      </halt>
    </application>
```

In this case, an application is launched on *machineA* using the script *sampleAppStart.sh* located in the specified directory. Additional login credentials may be specified, if necessary, with pre-set Secure Shell (SSH) keys. No behavior is specified for the initialize and execute states of the application.

2.4 Processes and Services

The Coordinator Architecture provides two primary components to implement its functionality. The central Coordinator process is the entry point to the Coordinator and is launched directly by the user. It is responsible for generating the SCXML, providing configuration information, executing the Simulation Execution SCXML and managing the Agents. The Agent process is responsible for interacting with a controlled process and executing a component SCXML. Each Agent corresponds to a single component. The Coordinator and Agents interact through a RESTful interface and SCXML events (Fielding 2000). Figure 4 shows a diagram of the primary processes and responsibilities.

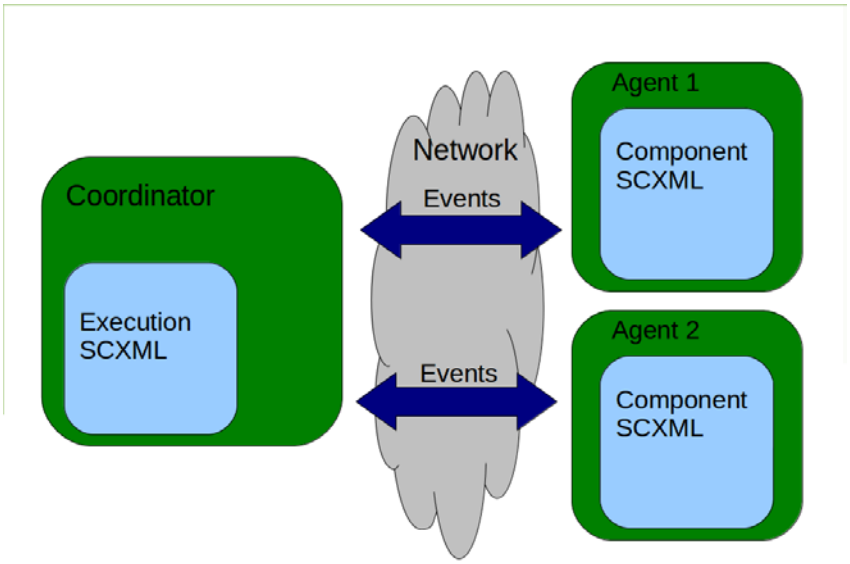

Figure 4: Coordinator and agent processes

The Coordinator provides the aforementioned RESTful interface for Agents to access configuration information and files. The Agent process, upon startup, makes a request for the list of configuration files it needs from the Coordinator. The Coordinator maintains a list of files for each Agent stored by a unique Agent Identification that is specified in the XML provided to the Coordinator by the Deployment Management System, and provided to the Agent as an argument when it is launched. The Agent then does GETs for each file required. Once all the files are received, the Agent opens a websocket to the Coordinator. The websocket is both persistent and bidirectional and used to exchange state machine events between the Coordinator and the Agent. This removes the need for the Agent to listen on an endpoint to receive events from the Coordinator, simplifying the host firewall configuration. Once the websocket is open, the Agent starts its state machine, which launches and controls the participant.

The primary responsibility of the Agent is to execute the SCXML for a component for control of a process. The Agent monitors the status of the controlled process as part of its execution. Having the Agent as a separate process from the central Coordinator allows the Agent to be collocated with the controlled process, allowing for greater level of interaction between the Agent and process, including Graphical User Interface (GUI) control. The Agent libraries contain all dependencies for running and controlling processes, which reduces the need to install additional packages on the host.

2.5 Process Control

One of the problems encountered in automating a simulation execution is how to interact with the processes and applications involved. Processes may be as simple as executing a launch command or may require additional console or GUI interaction on the part of the user. The capabilities for scripting process interaction will dictate the level to which the simulation can be automated.

2.5.1 Shell and Batch Scripts via SSH

A straightforward mechanism to control and automate processes is through shell or batch scripts. Developers often provide scripts as a simple mechanism to launch applications with specific configurations. The Agent provides the ability to directly invoke scripts at the desired state for a given component.

The scripts are invoked through SSH, allowing the command to be run on a remote host from the associated Agent for the component. We provide this capability using Java Secure Channel, which is a java implementation of SSH2 (JSCH 2012). The SSH launch command captures the Process Identifier (PID) of the executing process. This PID is then used to periodically status the process to assure that it is still executing. The status allows the Agent to detect an unexpected process termination and generate error event as appropriate.

Computers running Windows do not natively support SSH logon and process execution. To provide this feature, the PsTools toolkit from Microsoft is used by scripts to start and status the processes. It also allows processes started remotely to run with a GUI and be interactive. In addition, a freeSSHd SSH server is installed to support the SSH functionality and PsTools to capture running PID.

The Coordinator also uses the SSH command execution capability for launching the Agent processes themselves. The Agent process may then execute on a remote host from the central Coordinator process and be collocated with the controlled process. This collocation becomes important when performing more sophisticated interaction with the controlled process.

2.5.2 Graphical Control and Jython

Many processes that need to be controlled by the system require interaction with GUI components. A novel tool which provides this interaction is Sikuli Script, which automates interaction with a GUI and "anything you see on the screen" (Sikuli 2010). Sikuli is open source software under the Massachusetts Institute of Technology (MIT) license. It uses Jython, python for the Java platform, as the scripting language. Through a combination of Image Recognition and Optical Character Recognition (OCR), it allows users to script interactions, such a mouse clicks, drag and drop, entering text and changing window focus. One or more windows may be interacted with at a time. The Sikuli project also provides an Integrated Development Environment (IDE) for designing scripts.

The Coordinator leverages Sikuli Script by providing the ability to script control of an application using Jython. Users create a class within a module that implements an interface for executing the various states of the component. Jython not only allows python scripting within the Java platform, it also provides the ability to leverage Java libraries from within those scripts (Juneau et al. 2010). This design allows users additional ways to interact with a process, including ExpectJ, a Java implementation of Expect for interacting with console processes. In order for a Jython script to be used for process interaction, the Agent process must run on the same host and desktop as the controlled process.

3 USE CASES AND LESSONS IDENTIFIED

3.1 Cloud

The EASE system supports cloud based execution (Allen, et.al. 2012). Users can leverage virtualized computing and network assets to instantiate a simulation exercise, varying the application combination, configuration values and number of runs without end-user intervention.

Currently a cloud execution of EASE is available through a web interface. Users are able to use the EASE interview system to select and compose a simulation execution. In this use case, the Deployment Management System provides the Coordinator the execution sequence for the simulation and the number of iterations to perform. For example, the scenario used for the latest EASE Hands On Training class consisted of a OneSAF application, four Battle Command Management Systems (BCMS) applications, an Advanced Testing Capability (ATC) test case, and the supporting RTI environment process (Metevier et al. 2009). The Coordinator launches the simulation and provides status and state updates as the simulation progresses. Execution continues until a pre-described termination point, which in the example was a predetermined time limit. The user also has the options to shut down the current run manually or abort the execution. Upon shutdown, the Coordinator closes down all processes and executes the data collection routines. The EASE interview system provides the data artifacts back to the user and archives them for later reference.

In this use case, the coordinator improved the ability of the EASE system to monitor the progression of the simulation by receiving information of the current state of execution. In addition, certain errors, such as an application terminating unexpectedly, could trigger an event to stop the execution and provide information on the failure back to the user sooner than waiting for the execution time period to elapse. Further, since shutdown of the simulation is represented as a separate state and can be transitioned through external events, adding the ability for the user to manually shutdown as an external event was straight forward.

An issue with the current implementation is the level of detail provided in status messages. While they provide information on the state of the simulation or if a particular application fails, they are not directly linked to further data on the error or application. This situation somewhat hampers the ability to address the error efficiently. In turn, improved error diagnostic and handling are part of the future development on the project.

3.2 Simulation Lab

In addition to running within a fully virtual environment, the EASE prototype was installed at MSCoE where there is dedicated hardware with preloaded simulations for executing scenarios used to support its analytical mission. The supporting infrastructure of hardware, operating systems and networks are determined by the scope of the scenario and the simulations required to support these scenarios. The MSCoE is also constrained by Department of Defense (DoD) Information Assurance Certification and Accreditation Process (DIACAP) rules, which determine available services and operating systems present on the hardware (DoD 2007).

The goal of the deployment to MSCoE was to improve operations at the lab in support of its exercises and experimentation. Specifically, EASE will support the MSCoE portion of the BLCSE SimEx '13 experiment. As discussed in section 2.1, this entails three clusters of OneSAF simulations, along with an application for connecting to the large HLA federation. In this use case, the lab support personnel would use the EASE system to manage the simulation environment. The Coordinator, as part of the EASE installation, would be responsible for launching and performing initialization of the simulations. The support personnel would be responsible for determining when to shut down the simulation. The Coordinator would then perform the data collection tasks.

The EASE system was initially deployed in late February 2013 at MSCoE and later updated in June 2013. The support personnel were trained in the use of the system, which was demonstrated to run a small exercise consisting of a set of OneSAF applications and a countermine application in an HLA environment. The Coordinator was able to run the desired applications and repeat the execution easily as desired. The support personnel maintained control of how long the execution ran. Any startup errors were quickly recognized and reported, such as one of the hardware assets being unavailable for a run.

The experience supporting MSCoE called out the need to simplify the installation and configuration of the EASE system, including the Coordinator, in an existing laboratory environment. In addition, the

ability to run the Coordinator as a standalone piece can be improved to allow users to leverage its capabilities to manage an execution outside the EASE system.

Another shortcoming of the current implementation relates to the options for handling error conditions. The Coordinator allows the failure of an application to be ignored or to cause the execution to shut down. Another option of automatically restarting the process should also be supported. A possibility for this capability would be to introduce error recovery states for the execution state machine as well as for the applications. This mechanism should allow the Systems Engineer to specify how to recover from the failure and is an area for future research.

4 CONCLUSION

The Coordinator has demonstrated an improved execution management over previous automated approaches and manual operations. Much improvement is attributed to the advanced workflow specification that can be supported by State Chart representation of the execution as well as the inherit ability to react to internal and external events. The event mechanism can be used in fully automated simulations to use more advanced criteria for an execution to move to the next stage or completion, beyond simple time sequencing. The event mechanism also allows for live user control of simulation progression. An area of development will be to determine how to add better fault recovery into the state charts for managing the simulation and its components.

The next deployment of the EASE system will be in the simulation laboratory of the United States Military Academy (USMA) Department of Systems Engineering. This deployment will leverage more Virtual Machines for the management system and simulation applications, simplifying the installation process. While the scale of simulation scenarios for USMA is smaller than MSCoE, the experiments are more time constrained. In some instances, a scenario needs to be run as part of a 50 minute class. The scenarios are currently manually executed and managed, which can consume much of the class time. The execution is further complicated by the fact that the composition and configuration of a scenario can change based on objectives. The goal of the EASE system will be to simplify use of simulation within this teaching environment, allowing more focus on the objectives as opposed to the mechanics of the simulation exercise.

REFERENCES

Allen, G. W., S. Gallant, C. Gaughan, L. and Schroeder. 2012. "Vision of a Composable Architecture Simulation Environment (CASE)." *Spring Simulation Interoperability Workshop 2012*.

Beauchat, T. A, S. Gallant, C. Gaughan, and C. J. Metevier. 2012. "A Collaborative Tool for Capturing the Design of a Distributed Simulation Architecture for Composable Execution." *Spring Simulation Interoperability Workshop 2012.*

Department of Defense Instruction Number 8510.01. *DoD Information Assurance Certification and Accreditation Process (DIACAP).* November 28, 2007. Washington, DC.

Fielding, R. T. 2000. "Architectural Styles and the Design of Network-based Software Architectures." Doctoral dissertation, University of California, Irvine.

Gallant, S., C. J. Metevier, and C. Gaughan. 2011. "Systems Engineering an Executable Architecture for M&S." *Fall Simulation Interoperability Workshop 2011.*

Gallogly, J., S. Gallant, C. Gaughan, and H. Marshall, Henry. 2012. "Programmatically Identifying Composable Simulation Capabilities." *Spring Simulation Interoperability Workshop 2012.*

Harel, D. 1987. "Statecharts: A visual formalism for complex systems". *Science of Computer Programming*, 8(3): 231–274.

Hurt, T., J. McDonnell, and T. McKelvy. .2006. "The modeling architecture for technology, research, and experimentation." In *Proceedings of the 2006 Winter Simulation Conference*, Edited by L. F. Perrone,

F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 1261-1265. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Institute of Electrical and Electronics Engineers. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules. 2000.

Juneau, J., J. Baker, V. Ng, L. Soto, and F. Wierzbicki. 2010. *The Definitive Guide to Jython*. version 1.0. Creative Commons 3.0. http://www.jython.org/jythonbook/en/1.0.

Metevier, C. J., C. Gaughan, S. Gallant, L. McGlynn, J. McDonnell, G. Smith, and K. Snively. .2009. "Modeling Architecture for Technology Research and Experimentation (MATREX): M&S Tools and Resources Enabling Critical Analyses". *MSIAC Journal*, July 2009.

Murphy, S., M. Diego, and S. Gallant, Scott. 2011. "U.S. Army Modeling and Simulation Executable Architecture Deployment Cloud Using Virtualization Technologies to Provide an Extensible Platform as a Service (PaaS) Cloud for Federated Application Configuration and Execution." *Fall Simulation Interoperability Workshop 2011.*

Object Management Group. 2009. "OMG Unified Modeling Language (OMG UML), Infrastructure Version 2.2." http://www.omg.org/spec/UML/2.2/Superstructure/PDF.

Powell, E. T., and J. R. Noseworthy. 2012. "The Test and Training Enabling Architecture (TENA)" In *Engineering Principles of Combat Modeling and Distributed Simulation*, Edited by A. Tolk, 449-477. Hoboken, NJ: Wiley.

Sikuli Lab. 2010 "Sikuli Script". University of Colorado. http://www.sikuli.org.

Terracotta, Inc. 2013. "Quartz Overview." Quartz Scheduler. http://quartz-scheduler.org/overview.

World Wide Web Consortium. 2012. "State Chart XML (SCXML): State Machine Notation for Control Abstraction." http://www.w3.org/TR/scxml/

World Wide Web Consortium. 2007. "XSL Transformations (XSLT)Version 2.0." http://www.w3.org/TR/xslt20.

Wittman Jr, R. L., and C. T. Harrison. 2001. *OneSAF: A product line approach to simulation development.* Technical Report, The MITRE Corporation, Orlanda, FL.

AUTHOR BIOGRAPHIES

KEITH SNIVELY is a Principal Software Engineer with Dynamic Animation Systems, Inc. He has over fifteen years of experience in distributed computing and Modeling and Simulation for the DoD. He has worked on design and development of several distributed communication systems, including RTI-NG and the TENA middleware and also participated in the development of the HLA 1516 Evolved specification. Currently, Mr. Snively serves as a principal software developer in support of the U.S. Army Research Laboratory (ARL) Human Research & Engineering Directorate (HRED) Simulation & Training Technology Center (STTC) Advanced Simulation Branch and specifically supports design and development of the MATREX RTI-NG, ProtoCore and EASE. Mr. Snively received an M.S. degree in Mathematics from the University of Virginia. His e-mail address is ksnively@d-a-s.com.

RICHARD LESLIE is a Principal Systems Engineer with Kinex, Inc. He has over 35 years of experience with modeling and simulation supporting various US Government programs working in technical and managerial positions. Mr. Leslie received his MS degree in Computer Science from The College of William and Mary and a BS in Physics, Mathematics, and Computer Science from Heidelberg University. His e-mail address is rleslie@kinex.com.

CHRIS GAUGHAN is the Chief Engineer for Advanced Simulation and Deputy Technology Program Manager of the Modeling Architecture for Technology, Research and Experimentation (MATREX) program at the U.S. Army Research Laboratory (ARL) Human Research and Engineering Directorate (HRED) Simulation and Training Technology Center (STTC). He has a diverse portfolio of distributed simulation projects that support the full spectrum of the Department of Defense Acquisition Life Cycle.

He received his Master of Science and Bachelor of Science in Electrical Engineering from Drexel University in Philadelphia, PA. His e-mail address is OPS.STTC@peostri.army.mil.