Executable Architecture for Systems Engineering
– Distributed Modeling Framework

Colonel Robert H. Kewley
Department of Systems Engineering
United States Military Academy
West Point, NY 10996
845-446-5534
Robert.Kewley@usma.edu

Major Stephen J. Sapol
Department of Systems Engineering
United States Military Academy
West Point, NY 10996
845-446-3573
Stephen.Sapol@usma.edu

Keywords:
DEVS, Systems Engineering, Federated Modeling

ABSTRACT:  The growth of model based systems engineering within the systems engineering community is resulting in a call for the ability of the system architecture to be able to directly call a simulation model for system evaluation. System functions, value properties, and configurations are directly sent to the simulation as independent variables, and system performance metrics are dependent variables. The Executable Architecture for Systems Engineering (EASE) provides a user interface for the execution of simulation experiments in a set of cloud-based computing resources. However, implementations of these simulations in the High Level Architecture (HLA) or Distributed Interactive Simulation (DIS) are challenging. It is particularly difficult to identify simulation parameters that should be mapped to architecture properties because these simulations have their own instances of initialization data, state data, events, time advance, and output data. The federation must coordinate these individual copies to form a coherent scenario run. This coordination and management task is resource intensive for any realistically complex scenario. The EASE Distributed Modeling Framework (EASE-DMF) provides a different way to build federations. In order to compose models, the developer must first break them apart from their containing simulations— assembling a set of models required by the systems of interest. Each component is a Discrete Event Systems Specification (DEVS) atomic model of a system function. Computations are extracted from existing simulation models as pure state transition functions with no side effects. The framework provides the ability to manage initialization properties, state, scheduling, and parallel execution. Each component exposes its initialization properties, state, and event message interfaces via Extensible Markup Language (XML) in order to enable code generation and to support identification of parameters that can be linked to system architecture properties or to performance metrics. EASE-DMF guides the development of models, the integration via event messages, and the execution in support of an experimental design. A reference implementation of EASE-DMF uses the actor model of computation within the Akka actor framework. This framework has a natural alignment with the message-based nature of DEVS, and it has built in support for asynchronous computation, distribution of models across networks, and parallel execution. A model scenario integrates existing target acquisition, communication, and terrain models in a simulation of soldier situation awareness. It supports evaluation of soldier command and control architectures.
1. Introduction

Consider a situation in which a systems engineer is developing the systems architecture for an advanced squad-level command and control system. This hybrid formation employs both humans and robots. One architectural decision is the selection of a radio and sensor package for an unmanned ground vehicle (UGV). A SysML block definition diagram for this system is shown in Figure 1. Because the vehicle has limited carrying capacity and power, a trade decision must be made. In general, better sensors and radios take up more space and consume more power, so their performance is capacity limited. These constraints can be handled with very simple calculations. However, system performance evaluation requires more complex analysis. Suppose we wanted to maximize the target acquisitions of a system in a simple scenario. Maximizing sensor capability would allow the system to acquire targets, but it would potentially limit the abilities of the radio, so those targets could not be reported to the squad. The designers must achieve a proper balance between sensor and radio performance to maximize situation awareness in a variety of enemy, terrain, and weather scenarios. This requires a well-designed simulation experiment.

![Figure 1: Block Definition Diagram for Squad Command and Control System](image)

It is difficult to find a single simulation model that has the required fidelity of sensor modeling and communications modeling. Even more complex considerations, such as UGV mobility, bring in additional modeling requirements. Engineering trade studies with even moderate levels of complexity provide a challenge for performance modeling. We are often forced to design an experiment using federated solutions where individual models must be run in synchronization to answer system performance questions. The Executable Architecture for Systems Engineering – Distributed Modeling Framework (EASE-DMF) research effort is working to develop approaches that enable the performance analysis of systems by integrating models from the engineering domains that correspond to the system design.

1.1 Challenges of Federated Simulation Systems

Current practices for federated simulation make the above scenario very difficult. Distributed Interactive Simulation and High-Level Architecture are integration technologies that only share minimal state data [1-2]. Simulation initialization, simulation state, time, terrain, and models must be coordinated across federates. Coordinated simulation initialization is complex. Input data is hidden in simulation specific data files. Given any complex scenario with more than two or three federates, analysis is painstaking and fraught with errors. Verification and validation of the federation is an immense challenge [3]. These challenges are a direct result of the decision to federate entire simulations, vice discrete event computations. Many different labs and acquisition programs in the military community have domain-specific simulation models to replicate and analyze systems of interest. Re-use of those models provides cost savings over building a new model combining their capabilities. However, as complexity of the system grows, management and execution of this federation becomes ever more difficult [4]. While some methodologies for federation development in system of systems engineering have been developed, they require extensive engineering to deal with and synchronize disparate entities, information exchange systems, models, environmental representations, output data, and time [5]. These approaches result in federations that cannot be used by ordinary engineers, analysts, or training developers. They require simulation professionals at every step of the way.

2. Overview

2.1 Executable Architecture for Systems Engineering (EASE)

The Executable Architecture for Systems Engineering
(EASE) provides a user interface for the execution of simulation experiments in a set of cloud-based computing resources. EASE allows for the rapid coordination and execution of simulations by engineers and M&S users based on functional requirements determined through an interview process.

EASE provides users the ability to develop simulations based on the requirements determined in the interview process. The capabilities of available simulations are determined beforehand. In order to run these simulations, there may be a requirement to federate simulations to meet the requirements of the experiment. EASE also captures the interoperability requirements between simulations. Once the requirements are established and scenario is created, EASE will configure the software as necessary and allow for the simulations to be run. EASE also is responsible for the execution and time management within the simulation. [4]

2.2 Executable Architecture for Systems Engineering – Distributed Modeling Framework (EASE-DMF)

The Executable Architecture for Systems Engineering – Distributed Modeling Framework (EASE-DMF) provides the tools for the simulation analyst and engineer to build scenarios, combine models, set properties and collect data in order to analyze system performance. It allows for the modular construction of a series of discrete event models. These models are constructed as state transition functions, where inputs are provided to the model in the form of properties, state, and time. Each model provides predictable outputs in the form of state changes and new events. These models are loosely coupled, in a parallel and distributed manner, while time and schedules are centrally managed by the framework. This allows for the reuse of data definitions and functions.

The high-level goals of the EASE-DMF infrastructure are the following:

- Enables efficient design and execution of experiments.
- Allows parallel computation for execution of large-scale experiments.
- From the standpoint of simulation development and execution, EASE-DMF allows a systems engineer and simulation analyst, to construct a simulation utilizing the modular sub-system models. This requires the development of stand-alone, discrete event sub-system models. Once the simulation is developed and executed, the data input, output, and analysis will be integrated as opposed to the multiple formats currently utilized in federated simulations.

2.2.1 Discrete Event System Specification (DEVS)

DEVS models are useful because they are modular models, which can be constructed in a hierarchical manner. These modular models can run parallel to one another, which allow for the coupling of models [6]. An example of a potential simulation using DEVS is a platoon on platoon combat scenario shown below in Figure 2.

![Figure 2: Platoon Combat Model](image)

The entire model decomposes into two coupled platoon models in opposition to one another. Each platoon decomposes into coupled, individual soldier models. Each soldier model then decomposes into the basic atomic models: Detection, Maneuver, Engagement_Atk, and Engagement_Dmg. Within a DEVS coupled model, “the combat scenario and the combat entity structure, and the DEVS atomic model in combat modeling is the entity that is not decomposable any further. [7]” Figure 2 also demonstrates the relationships between models and the relationship of the inputs/outputs of each.

2.2.2 Base Object Model (BOM)

BOMs allow for the common representation of objects
throughout the simulation. This allows for interoperability, reuse, and composability of models and simulations [8]. The BOM allows for model identification, the development of a conceptual model, model mapping class definition, and the characterization of data types [9]. The utilization of Base Object Models in EASE-DMF is critical in order to add semantics of the model, to specify the inputs and outputs of models, and to situate the models in a chain of interactions.

2.2.3 Parallel Algorithms

Due to the distributed nature of federations and the increased use of cloud-based computing, EASE-DMF requires the ability to run parallel algorithms. Parallel algorithms allow for multiple operations to be performed at one time. Ideally, parallel algorithms lead to more cost effective computing solutions [10]. This is vital when attempting to run a distributed, federated simulation. EASE-DMF will utilize a method of concurrent simulation called Time Warp Simulation. Time Warp Simulation allows an object to progress on its own simulation time, and rollback to the past when required. This will be driven by when a model receives a message from another object that is lagging behind. The model that is ahead will revert back to the time of the lagging model it received a message from. The benefits of Time Warp Simulation are that it is transparent to the programmer, it can simulate any discrete model, it is independent of processors or network, and it cannot deadlock. [11]

3. DEVS Model Implementation

In order to implement EASE-DMF, a basic force on force scenario was developed. The base scenario will be the one provided in the introduction of this paper. We will have a UGV move towards and locate an enemy soldier. The UGV will report the enemy location to a friendly soldier and initiate a movement order. The friendly soldier will move towards and engage the enemy. Ultimately, the initial scenario is not to develop a robust, new simulation, but to test the ability of the framework to federate models with relative ease. Due to this, we decided on a simple scenario to build in basic functionality. The scenario will be constructed in three phases, each building on one another.

The initial scenario has a friendly soldier move towards the enemy. Once the enemy is in the friendly field of view and is acquired, the friendly soldier will determine if it can engage the target. If not it will move towards the target until it can engage, and engages the target. The enemy will be static for this run. This will allow for the testing of multiple models. These include the movement model, acquisition model, and engagement model.

The next scenario will introduce the UGV into the scenario. Here we will have one friendly UGV, one friendly soldier, and a static enemy soldier. In order to construct the UGV entity, we can incorporate our target acquisition model used for the soldier with a new movement model and a communications model. This will allow us to test the modularity concept of EASE-DMF as well as integrate new communications and behavior models.

The final scenario will consist of a friendly soldier, a friendly UGV and the enemy soldier is no longer static. The friendly soldier already constructed will be modified in order to create an enemy soldier with different properties. This will require modifications to sensor, weapon and behavior models. These changes create a more complex and dynamic scenario to further test the ability of the EASE-DMF to manage and execute simulations.

In addition to the soldier models constructed, a terrain model will be used to interact with the entities on the battlefield. Although these are simple scenarios, they will allow for multiple atomic models to be pieced together, utilizing EASE-DMF to manage the simulation. An overview of the simulation is shown below in Figure 3.

This scenario can eventually be constructed to conduct the simulation in order to make the trade decision discussed in the introduction of this paper. The parametric diagram provided in Figure 4 provides an overview of the proposed simulation. Ultimately, the simulation wants to determine the number of targets reported to the Friendly Squad. This is a function of the number of acquisitions the sensor makes. The DEVS target acquisition model, utilizing the Army Material Systems Analysis Activity’s (AMSAA) ACQUIRE methodology [12], will be used to determine if targets are acquired. The ability of the squad to receive the acquisitions is dependent upon successful radio transmissions. The target acquisition model will be discussed in more detail in a following section. The
DEVS communications model, using the Communications Planner for Operational and Simulation Effects with Realism (COMPOSER) methodology [13], will take into account the radio properties and terrain to determine success or failure of each transmission. The terrain model will use the Layered Terrain Format to store terrain data and provide efficient computations of terrain interactions such as line of sight or presented area of the target [14]. The output of these models will be utilized by the coupled Situational Awareness Simulation in order to determine the number of targets reported.

EASE-DMF will allow for designers to work with simulation experts to design the simulation, combining the required DEVS models. It will enable the engineer and analyst to develop a well designed experiment in order to determine the optimal balance between sensor and radio performance to allow the squad to identify targets by varying both sensor and radio parameters.

![Figure 4: SysML Parametric Diagram of Trade Study Simulation](image)
3.1 Base Models

In order to build this scenario, a series of unique models must be developed. A basic representation of the coupled soldier model is provided below in Figure 5.

This model is a single coupled soldier model showing additional attributes and atomic sub-models which all interact. This model becomes even more complex when incorporating a friendly soldier model, an enemy soldier model, and the terrain interface. This is shown in Figure 6.

EASE-DMF utilizes a series of DEVS models. An output of one model will act as an input to another model. Mutable states are centrally updated by the framework. EASE-DMF maintains and stores both the mutable states and static properties. Due to this only the simulation events are distributed. Developers of these atomic models will only need to be concerned with inputs from other models and the centrally managed state and properties required. An example of this is the Acquire model described below.

3.1.1 ACQUIRE Model

The first DEVS model developed utilizes the ACQUIRE methodology from AMSAA. This model is used to determine a soldier is able to acquire a target. The acquire model receives its inputs from three different sources. Annotated XML is used to publish and pass information.

Other sub-models within the larger model are the first source of inputs for the Acquire model. For example, the Enemy Soldier Model will provide the target location input for the Acquire Model. The Acquire methodology utilizes weather data as an input. The initial scenario will set weather as a static property. It will be maintained and stored by the framework and will not change. If a weather model is constructed, it could be easily integrated into this scenario utilizing EASE-DMF by mapping its outputs to the required inputs to the Acquire model. The final source of inputs for the ACQUIRE model is the internal state, again centrally updated, managed, and stored within the framework. This information can include information such as orientation and location.

Utilizing functional programming for the external state transition function, the ACQUIRE model receives these inputs and a predictable series of outputs is provided. The same series of inputs will always produce the same outputs. This output is then used in another, independent DEVS model elsewhere. For example, the engagement model will receive the output of the ACQUIRE model as an input target for the engagement process. A graphical representation is provided below in Figure 7.
4. Technical Implementation

The technical implementation of EASE-DMF takes advantage of recent advances in computer science that enable this sort of distributed reactive development. These include reactive programming, the Akka actor framework for concurrent distributed applications, the Scala programming language, and the Java Architecture for XML Binding.

4.1 Reactive Programming

The emergence of high demand real-time applications on the web and in mobile computing environments has challenged developers to build systems that can react and respond in this environment. A team of experienced developers has authored the Reactive Manifesto to describe four necessary high-level system requirements [15].

**Event-driven** applications built on asynchronous communication separate the details of implementing reactions from the communications system that connects the distributed components. This approach yields loosely coupled components that are easier to maintain and extend. This event-driven principle closely aligns with the loosely coupled models of the DEVS framework that respond to external simulation events. It also lends itself nicely to parallel computation.

**Scalable** applications allow the addition and distribution of computing resources while minimizing the changes needed in the application design. This location transparency provides a wide range of options for responding to increased demand. These include adding processors with more threads, distributing across computers, using massively parallel super-computers, or deploying to cloud-based or even mobile resources. This location and hardware independence has always been one of the key tenets of simulation integration frameworks. Reactive applications provide scalability in a more natural way.

**Resilient** applications design appropriate responses to failure into each component. This model allows failures to be captured, isolated, and dealt with while minimizing the impact on the rest of the system. Distributed simulation frameworks have a similar need for responding to failures as nodes join or leave the federation due to issues with computers, networks, software, or users.

**Responsive** applications provide rich and timely interaction even in times of high traffic or network restriction. Key elements of responsive design include storing data close to where it is used, reacting quickly as data changes, and managing the overhead and network loading associated with data and message exchanges. These principles will allow a distributed simulation to run efficiently – even under heavy load.

4.2 Actor Model implementation

The actor model of computation is one way to deal with concurrency [16]. In short, an actor is a computational agent that responds to each incoming message with a set of messages sent to other actors, a new behavior which will influence the response to future messages, and the creation of new actors. The Akka framework is an element of Typesafe’s Reactive Platform that is designed to provide an implementation of the actor model, fault tolerance, location transparency, and persistence [17].

The Akka actor framework is an ideal environment for development of EASE-DMF because of its full support of reactive programming principles and the associated close alignment with the DEVS formalism. The EASE-DMF infrastructure is a set of Akka actors and associated components to allow parallel and distributed execution of DEVS models.

The SimActor is the main component of the infrastructure. Every DEVS atomic model is a SimActor. This component performs many of the important data, schedule, and time management functions necessary to execute the simulation. In order to execute a DEVS state transition function, the SimActor receives and responds to external messages or to internally scheduled events. An EventHandler interface is provided to developers of DEVS models that exposes all of the actor’s internal state variables, along with the state other actors as recorded by messages sent to the SimActor via subscription. In addition, the static properties of the actor are exposed. The SimActor has in internal schedule that it steps through as time moves forward. As a result of state transition computations, the SimActor will update internal state, schedule internal events, and send event messages to other SimActors.

The framework generates random variates in a coordinated and distributed fashion. A global random engine defines the parameters of a linear congruential generator. Each SimActor periodically gets a set of random variates, perhaps a 100 of them, from the global generator. The actual values of the variates are not passed, just the seed value. Each SimActor will be able to use the seed to calculate the next 100 variates. It then sends a message to the global generator to get the seed for the next 100. The global generator will increment its seed 100 times to prepare for the next request. When a SimActor performs a state transition function, it will receive the necessary number of random variates as input.
This architecture maintains the referential transparency of state transition functions. That is, a referentially transparent function can be replaced in an expression with its value, and the expressions value will be the same. Stated another way, given a defined set of inputs, a state transition function will always produce the same result. It is deterministic, without any side effects such as updating state or reading data from sources other than the input. The property is important in order to manage the parallel and independent execution of simulation events.

The SimActor also performs all of the necessary time and state management functions in order to implement the time warp parallel discrete event function algorithm. The main requirement is the ability of a DEVS model to roll back to a time in its past if it receives an event message that took place at an earlier time. This involves setting state variables to previous values, resetting the internal schedule to its contents from the previous time, and sending anti-messages to other actors that tell them to disregard messages that were sent after the rollback time. These messages can trigger a chain of rollbacks in other SimActors.

In order to create a DEVS atomic model, a developer would need to extend the SimActor. First, the developer must define an initial state of the actor using a series of state variables, each represented as a Java object. They would then have to define a set of event handling methods that respond to different types of simulation events. These event handlers use the actor’s internal state and properties, the data in the event message, and any random inputs to calculate a state transition. The event handler returns a result which contains internal state updates, events to schedule, and messages to send to other actors.

One goal of the implementation is to enable developers to re-use code written in a variety of languages. Because the implementation is written in Scala, it integrates nicely with Java, or any other languages written for the Java Virtual Machine. Integration with other languages can be achieved through other means. One method is the Java Native Interface [18]. Another option is the Apache Thrift Interface [19]. This provides integration with functions written in a number of other languages. The actor model allows encapsulation of functions in other languages in an actor interface that allows loosely coupled integration via messages and asynchronous execution.

4.3 Simulation Data Representation

Data for a DEVS model comes in three forms – static properties of models, dynamic state of models, and messages passed between models. The high-level design goals of EASE-DMF include the exposure of model input data so that it can be manipulated, the exposure of state data during simulation runs, and the documentation of messages passed between models. In order to achieve this, each static property and each state variable of a model must be represented as an element of an XML schema document. The same is true for data passed as messages. This requirement allows Java objects to store and pass data during simulation execution to be automatically generated via the Java Architecture for XML Binding (JAXB) [20]. This architecture leads to a number of advantages:

- Input data can be expressed and manipulated in XML using existing tools or web interfaces.
- State data can be serialized as XML files for ingestion by XML compliant databases and subsequent analysis.
- Records of message interactions can be recorded as XML.
- Many existing simulation systems already use XML for data input, so re-use of those structures by DEVS models is enabled.
- These specifications can be stored in the Base Object Model in order to thoroughly document the specification of a DEVS model.

Finally, messages in Akka must be immutable. While JAXB generates standard Java objects, there are plugins available that allow generation of immutable objects whose data elements are final and immutable [21].

5. Conclusions and Future Work

This initial phase of EASE-DMF implementation assesses whether the overarching design goals of model integration, exposure of input and state data, parallel computation, experimental design, and binding the simulation parameters to the systems architecture can be achieved. Transition the initial prototype into a robust infrastructure that supports military and non-military simulations. The robust infrastructure will add the following features:

- Visualization and query of state data and property data using adaptations of standard tools
- Certificate signing of property data and
functions as a means of security but also as a means of verifying and validating the data and computations by the appropriate authorities.

- Enable, through a technology such as event sourcing, the ability to re-create the simulation state at any time during the run.
- A graphical model and user interface to compose functions, events, and data into a working simulation.
- Parallel and distributed implementation for multi-core or cloud computing.
- Support for writing events or functions in selected standard languages.
- Support for implementing events or functions via network messaging.
- A user interface for design and analysis of experiments
- A capability for heuristic simulation optimization.

These capabilities will enable the analyst and engineer, with little or no support from software developers, to design and run distributed and parallel simulations to estimate the performance of complex systems to support design and trade decisions in the systems design process.

6. References


**Author Biographies**

**COLONEL ROBERT H. KEWLEY** is currently the head of the West Point Department of Systems Engineering. He is a career Army officer with a background in Armor. He is a graduate of the West Point Class of 1988 with a degree in mathematics. He holds a Master’s of Industrial Engineering and a Ph.D. in Decision Science and Engineering Systems, both from Rensselaer Polytechnic Institute. Colonel Kewley’s research interests focus on simulation and command and control systems.

**MAJOR STEPHEN J. SAPOL** is currently an instructor and analyst at the West Point Department of Systems Engineering. He is a career Army officer with a background in Field Artillery. He is a graduate of the West Point Class of 2004 with a degree in Operations Research. He holds a Master’s of Industrial Engineering from Georgia Tech and a Master’s of Engineering and Management from MIT. Major Sapol’s research interests focus on simulation and system dynamics.