A Proposed Open System Architecture for Modeling and Simulation (OSAMS)

A Service Oriented Architecture (SOA) for the M&S Community…

Jeffrey S. Steinman, Ph.D.

JPEO CBD
Software Support Activity
Integration and Test
Motivation…

• How would our lives improve if the cost of M&S was reduced by *an order of magnitude*?
  – How about *two orders of magnitude*?

• M&S provides a cost effective way (and sometimes the only way) to support many challenging applications
  – However, we believe that the true potential cost savings for M&S has not been realized!

• If we are going to reach these potential cost savings, we cannot continue doing things the same way
  – Current M&S interoperability standards are not adequate
  – Revolutionary, not evolutionary change is needed
Interoperability Standards

- Current interoperability standards allow simulations to interoperate
  - However, there are no standards for how to build models!

We should be building models, not simulations!

- OSAMS provides standards that specify how to build highly interoperable models
  - OSAMS-compliant simulation engine required to host models
  - OSAMS-compliant models must not deviate from the API
  - OSAMS is part of a bigger Standard Simulation Architecture (SSA) that has been carefully constructed to support interoperability with other standards such as HLA, DIS, TENA, and web-enabled SOAs
Some Basics… What is M&S

Model

A physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process.

DIS Glossary of M&S Terms, DoD Directive 5000.59, DoD Publication 5000.59-P and MSETT NAWC-TSD Glossary

Simulation

A method for implementing a model over time.

DoD Directive 5000.59 and DoD Publication 5000.59-P

Modeling & Simulation (M&S)

The use of models, including emulators, prototypes, simulators, and stimulators, either statically or over time, to develop data as a basis for making managerial or technical decisions. The terms "modeling" and "simulation" are often used interchangeably.

MSETT NAWC-TSD Glossary
From a Software Developer Perspective

• Terminology: Simulations
  – Simulations are programs that are composed of models
  – Simulations generally require a simulation engine to provide core event-scheduling and event-processing services that allow models to advance in time

• Terminology: Models
  – Models are software representations of systems
  – Models can be self contained and therefore be reusable if…
    • Independent of the simulation engine (does not coordinate the passage of time)
    • Independent of other models (does not directly invoke methods on other models)
    • No reliance on shared global variables (encapsulation)
  – But… More likely, models are tightly coupled to the simulation engine, other models, utility services, and global variables (in other words, models are generally not reusable)
Current Interoperability Strategy
Simulation-to-Simulation Interoperability

- Integrating entire simulations when only select models from each federate are required…
  - High integration costs
  - Expensive and clumsy to operate
  - Unavoidable performance and fidelity tradeoffs
The Proposed OSAMS Strategy
Model-to-Model Interoperability

- A better approach is to create model repositories/libraries that can be linked together to form a composable simulation…
  - Low integration costs
  - Easy to operate
  - High performance and fidelity
Modern Applications of Plug-In Composability

• Many modern applications support plug in strategies to support interoperability between components developed by different vendors
  – Web services (SOA)
  – Graphics art
  – Office productivity tools
  – Video games
  – Entertainment systems
  – Wireless networks
  – Music software
  – CBD sensors in NCES environment

• So why not provide a plug-in SOA approach to provide model interoperability?
Plug and Play for M&S

- Model interoperability is much more difficult than traditional plug in systems because different categories of models require different interfaces (and there are a lot of them…)
  - Requires **standardizing common types of interfaces** such as sensor detection and track data, communications, command and control, various representations of complex motion, human intelligence, rules of engagement, etc.
  - Requires a **plug-in strategy** that decouples highly interacting model components
  - Also requires **composability tool** that verifies associations with other models when they are plugged in

- Polymorphic and publish/subscribe data exchanging techniques provide decoupling between software modules while still promoting full model interoperability

- Potential timing issues between models in their interplay affects where models should reside in network environments
Simulation Engines

• What simulation engines provide

  - Simulation engines provide the **core event-processing infrastructure** and **language semantics** required to enable the development and execution of complex models. Simulation engines allow applications to **coordinate their processing activities in simulated time**, which can be synchronized to the wall clock for real-time systems, or unconstrained for as-fast-as-possible synthesis and data analysis runs.
Simulation Engine & Models

Wall of Separation

M&S Technology

Simulation Engine
(Focus on Services)

- Commonly used modeling constructs and software utilities
- All network-related operations automatically provided
- Capabilities and technology advances leveraged by all models
- Simulation engine implements an Application Programming Interface

CBD Model Representations

Overarching Models
(Focus on Models)

- Focus is on models, not infrastructure or bookkeeping
- Composable interoperability provided between models
- Form repository of models that can be reused
- Models plug into the simulation engine and use the API
Hardware Composability

- Hierarchically Composing a Federation onto Hardware
  - Federations are composed of networked federates
    - Milliseconds
  - Federates are composed of one or more machines
    - Less than a millisecond on local area networks
  - Machines are composed of processing nodes
    - Microseconds if using shared memory
  - Nodes are composed of threads
    - Nanoseconds for context switching between threads
  - Threads are composed of functions
    - Much less than a nanosecond for function or method calls

Performance spans more than six orders of magnitude
Must apply reasonable hardware composition strategy
Model Composability

- Hierarchically Composing a Federation from Models
  - Federations are composed of federates
    - Communication through RTI
  - Federates are composed of entities
    - Entities may reside on different processors
  - Entities are composed of components and Federation Objects
    - Components within an entity are on same processor
  - Components are composed of subcomponents and Federation Objects
    - Hierarchical composition is recursive

OSAMS provides the required APIs that support...

- Flexible hierarchical model component construction
- Modeling framework for scheduling events
- Abstract interfaces to support component interactions
- Distributed object abstractions to support network operation
- Data logging and trace file generation for debugging, analysis, and VV&A
Polymorphism Conceptualized

- **Old school** polymorphism was accomplished through class inheritance and virtual functions that are implemented by the derived classes. This approach is supported by all object oriented languages.
  - Inheritance required
  - Method names must match

- A more **modern** way to accomplish polymorphism is to define abstract interfaces that can be dynamically registered by class methods during run time. This is similar to the SOA methodology.
  - No inheritance required
  - Methods can be named anything
Architecture Rules for Model Interoperability

• Must preserve the abstraction that an entity may reside on any node when running in parallel, or within any federate when executing in an HLA federation
  – Entity state exchanged with other entities must be provided exclusively through Federation Objects
  – Entities interact with other entities exclusively through HLA-style Interactions

• Key to automating interoperability with HLA… Entities behave like miniature federates!
  – Entities are special SimObjs that are distributed to different nodes or federates when executing in parallel and/or distributed environments
  – Distributed object capabilities support HLA-like functionality between entities
  – Operator overloading in C++ can automate distribution of attributes
  – Interest management automatically operates on attributes
Trace File Generation and Time Line Analysis Tools
Summary of Composability Architecture
Rules and Properties

• Completely passive and encapsulated models with no relationships to other objects are automatically reusable
  – However, these kinds of models are rarely developed or openly shared

• To promote interoperability and reuse, all other models...
  – Must support a flexible *hierarchical composition structure* with the ability to define, compose, and construct simulation objects at run time
  – Must be allowed to advance time through services that are provided by a *standardized modeling framework* and compliant simulation engine
  – Must rely on *abstract polymorphic interfaces* to decouple interacting models
  – Must support *distributed object capabilities* to automate interoperability with legacy systems in a federated publish/subscribe environment and to support high performance computing
  – Must support *data logging interfaces and trace file* generation to support testing, debugging, analysis, and VV&A

An Open Standard Architecture for Modeling and Simulation (OSAMS) is required to promote model-based interoperability and reuse
How to Proceed with OSAMS

- **Phase I - OSAMS specifies all interfaces invoked by models**
  - Developers are required to implement the interfaces themselves within their own simulation engines (dependence on standalone utility libraries are ok)
  - Could support interface subsets as long as there exists at least one available simulation engine that supports all interfaces

- **Phase II - OSAMS provides common middleware software infrastructure with the right programming hooks to allow any simulation engine to implement the mapping**
  - Can significantly reduce costs of making a simulation engine OSAMS compliant
  - Requires development of the middleware capability
  - Potential technical issues involving the mapping

- **Phase III - OSAMS encourages the development of freely available open source compliant simulation engines**
  - Consolidates development costs, but has potential problems involving software rights, CM, industry buy-in, life cycle support, etc.
Summary & Conclusions

• Open Standard Architecture for Modeling & Simulation (OSAMS) is needed to lower the cost of M&S for CBD Overarching Models

✓ Strategy is not to just do things better… we must do things differently

In particular, reuse must begin at the model level

– This will lower costs of model development, VV&A, scenario generation, operation of simulation, post processing

– Better performance without compromising fidelity can be achieved by composing tightly interacting models together into a single executing process

– Next-generation capabilities can be achieved without throwing away investments in legacy simulations or M&S technology efforts
Final Thoughts

• OSAMS is a proposed SOA for the M&S Community and is based on proven technology and freely available open source software that could be used today

• OSAMS Specifically Addresses:
  – Plug and Play interoperability/composability of Models
  – Interoperability of Simulations