Rapid Synthesis of HLA-Based Heterogeneous Simulation: A Model-Based Integration Approach

Himanshu Neema
Harmon Nine
Graham Hemingway
Janos Sztpanovits
Gabor Karsai
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA
Email: { himanshu, hnine, heminggs, sztipaj, gabor } @isis.vanderbilt.edu

Abstract: Virtual evaluation of complex command and control concepts demands the use of heterogeneous simulation environments. Development challenges include how to integrate multiple simulation platforms with varying semantics and how to integrate simulation models and manage the complex interactions between them. While existing simulation frameworks may provide many of the required run-time services needed to coordinate among multiple simulation platforms, they lack an overarching integration approach that connects and relates the interoperability of heterogeneous domain models and their interactions. This paper outlines some of the challenges encountered in developing a command and control simulation environment and discusses our use of the GME meta-modeling tool-suite to create a model-based integration approach that allows for rapid synthesis of complex HLA-based simulation environments.

Keywords: Heterogeneous simulation, multi-paradigm modeling, model-based integration, meta-modeling, distributed simulation, high-level architecture.

1. Introduction

Evaluation of emerging command and control (C2) concepts necessitates a sophisticated modeling and simulation infrastructure that allows for the concurrent modeling, simulation and evaluation of (1) the C2 system architecture (advanced system-of-systems modeling), (2) the battle space environment (scenario modeling and generation), and the (3) human organizations and (group and individual) decision making processes (human performance and man-machine interaction modeling). Using simulated C2 environments to evaluate design concepts, validate new systems and components, and explore hazardous and ambiguous scenarios makes sense from both a cost and a practicality perspective. But, complex command and control environments have many disparate facets that need to be orchestrated, all of which cannot be handled by a single simulation platform. As a result, a federation of heterogeneous simulations, acting in a coordinated environment, must be employed.

Each individual simulation is composed of two parts: a domain-specific model and the underlying simulation engine. Additionally, each simulation engine, also called a simulation platform, may have its own unique execution semantics that need to be accounted for. All of the platforms and models in a federation must be coordinated in a meaningful way in order for the larger C2 simulation environment to be useful. Issues encountered in developing these environments relate to integrating simulations at both the platform and model levels.

Frameworks, such as the High-Level Architecture (HLA) ([1], [11], and [25]) provide APIs that have helped to reduce the complexity of integrating multiple different platforms, but many challenges remain in such environments. HLA, for example, does not provide any tools to design or deploy a federation. It primarily provides run-time support for various tasks, such as coordinated time evolution, message passing and shared object management. Up to this point, support for designing an integration model to tie together federates within a federation has been lacking. As a result, these frameworks require significant amounts of tedious, error
prone, and hand-coded platform integration code.

In this paper we present the C2 Wind Tunnel, a graphical tool environment for designing and deploying heterogeneous C2 simulation environments. Its primary contribution is to allow for the rapid development of “integration models”, and to utilize these models throughout the lifecycle of the simulated environment. An integration model defines all the interactions between federates and captures other design intent, such as simulation platform-specific parameters and deployment information. This information can be leveraged to streamline and automate much of the development and deployment processes.

The C2 Wind Tunnel environment uses the Generic Modeling Environment (GME) ([2] and [26]) tool-chain to construct the integration model. The GME is a meta-programmable visual modeling environment that allows designers to define domain-specific modeling languages (DSML) and to create and edit models using those DSMLs. The C2 Wind Tunnel team has developed a DSML tailored specifically to heterogeneous simulation model integration. Custom-written GME plug-ins, called “interpreters”, allow for the automated import of the salient portions of platform models directly into the integration model. The tool chain has been extended via additional custom interpreters to utilize the resulting integration model to automatically generate all necessary glue-code for each supported simulation platform and to manage the configuration, deployment and execution of scenarios throughout the simulation environment. Our custom extensions, combined with native GME capabilities, allow the C2 Wind Tunnel environment to be rapidly reconfigured and highly extensible, furthering the goal of rapid evaluation of emerging command and control concepts.

At run-time the Wind Tunnel leverages HLA as its framework for coordinating among federates. The HLA API manages lower-level coordination within the federation and simplifies what glue-code must be generated for each simulation platform. By providing a single comprehensive modeling and execution tool, the C2 Wind Tunnel enforces a holistic and rapid approach to designing and operating complex command and control environments.

The organization of the remainder of this paper is as follows. The next section discusses the background material and work related to our research. Section 3 details the C2 Wind Tunnel meta model and our approach for model integration using the GME environment. Section 4 reviews the details of the integration of several simulation platforms. Section 5 discusses our integrated approach to simulation deployment and execution. Section 6 covers results from using the framework in a real-world scenario. The final section concludes the paper and outlines planned future work.

2. Related Work and Background

There has been prior research performed on the integration of multiple diverse simulation packages, either with or without HLA. For example, one of the previous efforts at our institute, that relates to heterogeneous simulation, particularly of embedded systems, is the MILAN framework [10]. Efforts that relate to integrating simulation packages with HLA include OPNET [12], MATLAB-HLA [13], SLX [14], JavaGPSS [15], DEVSJAVA [16], [17], and PIOVRA [18]. As mentioned earlier, the HLA APIs provide run-time support but the problem of model integration is not addressed. Also, relevant commercial integration software does exist, such as the HLA Toolbox [19] for MATLAB federates by ForwardSim Inc. [20], MATLAB and Simulink interfaces to HLA and DIS based distributed simulation environments [21] by MÄK Technologies [22]. Additionally, there have also been some efforts on enhancing HLA support by complementary simulation methodologies such as in [23], and [24]. However, most of these efforts pursue HLA integration of isolated simulation tools, do not exploit HLA for distributed simulation, or are commercial applications for a dedicated simulation tool. Moreover, these efforts (except MILAN) do not have any support for model-based rapid integration of simulation tools, and limited-to-no support for automated deployment and execution of the resulting simulation environment.

2.1. HLA and GME Background

In contrast to prior research efforts, we have focused on developing a completely model-based integration approach. Our efforts leverage the GME tool suite for designing the
integration and deployment model and HLA for run-time support.

The High-Level Architecture is a standardized framework for distributed computer simulation systems. Communications between different federates is managed via the Run-Time Infrastructure (RTI) layer. The RTI provides a set of services such as time management, data distribution, message passing, and ownership management. Other components of the HLA standard are the Object Model Template (OMT) and the Federate Interface Specification (FIS).

The HLA standard focuses on three primary areas. First is time coordination throughout the federation. The evolution of time is a key thread through each of the integrated simulators, called “federates”. Each simulation platform must slave its progression of time to that of the overall HLA clock. The HLA standard provides several methods by which to accomplish this. Second is coordination of inter-federate messages and shared data objects. The HLA standard provides a publish and subscribe mechanism for passing messages and object updates throughout the federation. Third, the HLA standard provides for basic simulation execution control. Starting, pausing, and stopping the execution of a simulation is built directly into the HLA standard. The C2 Wind Tunnel relies upon all of these services during run-time.

Since HLA is an accepted standard, a number of commercial, academic, and alternate RTI implementations are available. Currently, we use the Portico RTI [4] – which provides support for both C++ and Java clients and is compliant with version 1.3 of the HLA standard.

The Generic Modeling Environment is a meta-programmable model-integrated computing (MIC) toolkit that supports the creation of rich domain-specific modeling and program synthesis environments. Configuration is accomplished through meta models, expressed as UML class diagrams, specifying the modeling paradigm (the domain-specific modeling language or DSML) of the application domain. DSMLs are able to capture both structural and behavioral aspects of a domain. Another way to envision this is that a DSML is a schema or data model for all the possible models that can be expressed by a language. Using finite state machines as an example, the DSML would consist of states and transitions. From these elements any state machine can be realized. The inherent flexibility and extensibility of the GME via meta models make it an ideal foundation for the C2 Wind Tunnel environment.

Alternate meta modeling frameworks, such as AToM³ [27], exist but extensive prior experience with the GME led to its selection for the C2 Wind Tunnel.

3. Model-Based Integration Environment

Complex command and control simulations require coordination between multiple heterogeneous simulation platforms. The HLA provides a standard for the RTI that supports the coordinated execution of distributed simulations. However, designing the model integration, coding the platform-to-RTI glue-code, and testing and deploying all of the various run-time components across multiple platform-specific simulation tools remains a challenging problem. Our project introduces a new approach for the simulation integration problem. The primary contribution of this effort is the development of a single holistic modeling and management environment, based on the GME, and a suite of model interpreters to coordinate between multiple platform-specific simulation tools.

A common problem with developing large-scale heterogeneous simulations is the complexity and effort required to integrate each domain-specific modeling tool into the larger environment. In the case of an HLA-based environment, not only does the RTI require a common federation definition (the .fed file), each involved simulation tool must also be integrated (via simulation platform-to-RTI glue code) and configured (in a platform-specific way) according to its role in the environment. Existing approaches treat the definition and creation of these artifacts as separate and not necessarily related steps.

The C2 Wind Tunnel environment automatically generates all of the required artifacts directly from the single, graphically composed, model in the central modeling environment. A large set of well-known simulation tools are supported and reusable run-time components are provided to integrate the various domain-specific platforms. As necessary, both glue code and configuration files are generated for each simulation tool.

3.1. C2 Wind Tunnel Meta Model

At the heart of the C2 Wind Tunnel environment is the overarching model integration DSML.
This meta model provides all of the modeling components necessary to specify the integration, deployment, and execution of the federated simulation models. Once the integration model has been defined for a given environment, a set of reusable model interpreters are executed to automatically generate platform-specific glue-code and all deployment and execution components. All generation and deployment steps directly rely upon the initial integration model.

Fig. 1 shows the primary portion of the meta model that defines the universe of composition elements. The three primary elements in a federation (defined by the model FOMSheet) are Interaction, Object, and Federate representing an HLA-Interaction, HLA-Object, and an HLA-Federate respectively. Note that proxy elements are simply references to their respective target model elements and they could be used in place of their targets. As defined by the HLA standard, federates in a federation communicate among each other using HLA-interactions and HLA-objects – which are in turn managed by the RTI.

Interactions and objects, in an analogy with inter-process communication, correspond to message passing and shared memory respectively. As seen in Fig. 1, the meta model fully supports the key HLA-defined attributes of these communication elements such as delivery method, message order (timestamp or receive order), and parameters. Also notable is that via the InteractionInheritance and ObjectInheritance connection elements, interactions and objects can form inheritance trees where children inherit the parameters or attributes respectively of parents. The ParameterType attribute on the Parameter and Attribute elements defines the data type of that element. The Attribute element also supports the HLA-defined attributes of Delivery and Order, which correspond to the desired delivery method of object updates and the type of timestamp updates are given.

Collectively, these meta model elements are required to define the relationships between all types of federates. Developing an integration model using these special simulation elements is discussed in the subsequent section.

The Federate element directly corresponds to any single instance of a simulation tool involved in the federation. The primary attribute of a federate, as far as HLA-based synchronization is

---

**Figure 1: Primary C2 Wind Tunnel Meta Model**
concerned, is its \textit{Lookahead} – the period of time in the future during which that federate guarantees that it will not send an interaction or update an object.

Lastly, the attributes of the \textit{FOMSheet} element capture the names and locations for configuration code that enables the integration of a range of supported simulation tools. We will describe this capability in detail in a following section.

Fig. 2 shows the additional portions of the C2 Wind Tunnel meta model. The top model in the figure defines how federates publish and subscribe interactions and object attributes. The middle portion of the model captures publish and subscribe links with Place elements – used for providing output from federates encapsulating a CPN Tools model. Similarly, the model at the bottom of the figure captures the connection with special-purpose \texttt{EndPoint} elements for the integration of OMNeT++ federates. CPN Tools and OMNeT++ are two of the simulation platforms integrated into the environment.

For obvious reasons, this metamodel, and its set of modeling elements, is very closely coupled to the HLA standard. With these elements a designer is able to completely specify the integration model of the entire federation and its constituent simulation platforms. Federates define the details of the platform-specific models that are involved, and their relationships are captured via publishing and subscribing to various interactions and objects.

3.2. Integrating Simulation Models

The semantic relationship between federates can be defined primarily using two main aspects: the data representation and the data flow. These are common elements of most of the domain-specific simulation modeling paradigms – so these are the key points of our integration models. The integration models describe both the data representation and data flow elements and, in some cases, include special elements as the placeholders for domain-specific models.

The data representation models consist of interaction and object models. Interactions are stateless and can have parameters while objects have states – which are represented as a set of attributes. As mentioned, both interactions and objects can have an inheritance hierarchy. These data representation models directly map to the HLA Federation Object Model (FOM).

Fig. 3 shows an example of two data representation models with an interaction class-
inheritance tree on the top, and an object class-inheritance tree on the bottom. All interactions must initially inherit from the InteractionRoot element, likewise all objects must inherit from the ObjectRoot element. This is done to both clarify the visual representation and to simplify interpreter transversal of the model tree. Deriving elements via inheritance is an intuitive approach readily understood by modelers.

Once the data representation models are created the modeler must define publish-subscribe data flow relations with federates. This is accomplished by connecting federates to interactions or object attributes with directional links. Federates publish and subscribe to any set of interactions or objects, dictated solely by the desired operational semantics. Federates can also publish or subscribe to entire data elements or to a subset of their attributes. Fig. 4 shows a simple data flow example specifying that the SensorFederate federate subscribes to the x, y, and z attributes of the Vehicle object class.

Integrating platform-specific models together in the central modeling environment is a simple matter of connecting federates to those interactions and objects with which they have a publish/subscribe relationship. This greatly simplifies the designer’s job since they no longer need to directly incorporate platform-specific considerations and can focus solely on the high-level interactions of the model. The lower-level integration details, such as clock management and message passing, are addressed once-and-for-all when the simulation platform is integrated into the general C2 Wind Tunnel environment.

### 4. Integrating Simulation Platforms

This section describes the process of integrating several example domain-specific simulation platforms into the overall C2 Wind Tunnel environment. For each one we outline how the engine aligns with the overall framework and the primary considerations involved in integration. The three example platforms are OMNeT++, Matlab/Simulink, and Colored Petri Nets (CPN Tools). This set was selected as representative samples of the typical types of platforms that are incorporated into command and control simulations.

Each integrated simulation platform has its own unique underlying execution semantics which directly impact the technical details of how it is integrated into the C2 Wind Tunnel environment. Each platform has a different model of computation and handles clock management and inter-simulation interaction events differently.

In addition to those simulation platforms covered below, the C2 Wind Tunnel environment currently supports DEVSJava [5], Delta3D [2], Google Earth, and C/C++ and Java-based custom federates. The set of supported platforms allows for the simulation of quite complex and diverse scenarios.

#### 4.1. OMNeT++ - Communications Federate

In a command and control simulation environment it is essential to model and simulate the communication network in order to study mission critical situations such as network failures or attacks. After evaluating multiple public domain network simulators, OMNeT++ [6] was selected as our network simulation platform. A primary advantage of OMNeT++ is its modular architecture which allows for the event-scheduler to be easily replaced, a requirement for HLA integration.

We developed a tool called NetworkSim - which is an HLA-compliant reusable communication network simulator based on OMNeT++. NetworkSim provides a set of high-level communication protocols (e.g. reliable send, streaming) while internally maintaining a faithful simulation of the full network stack. A key advantage of NetworkSim is its ability to utilize communications network models built using the standard OMNeT++ modeling tools. It simply handles the translation of messages from the RTI into appropriate network actions, and vice versa, and injects these messages onto the correct simulated network node. In addition to maintaining the underlying semantics of OMNeT++, this mechanism also serves to
isolate general RTI traffic from traffic on the simulated network.

A certain amount of RTI-to-NetworkSim glue code must be created. The code for all communications network nodes responsible for RTI communication, i.e. nodes appearing as end-points in the network topology inside of the OMNeT++ model, must be synthesized. A GME model interpreter reads the overall integration model and generates the C++ code needed for sending and receiving interactions and for publishing and receiving object attribute values via these nodes.

The generated glue-code builds upon the OMNeT++ API and interacts with the model-independent reusable code shown in the four boxes across the middle and with the OMNeT++ libraries shown at the bottom of the figure.

```c
Message *HLAScheduler::getNextEvent()
{
  Message *msg = sim->msgqueue.peekFirst();
  if (msg)
    throw new TerminationException(EXITOK);
  while (msg->arrivalTime() > rti->getTime())
  {
    rti->advanceTime();
    msg = sim->msgqueue.peekFirst();
  }
  return msg;
}
```

Figure 5: Extended OMNeT++ Scheduler

Apart from any domain-specific glue code necessary for RTI communication, NetworkSim also includes a reusable class that extends the basic OMNeT++ scheduler. This is necessary to synchronize the OMNeT++ simulation clock with the rest of the federation. Fig. 5 shows the key scheduler function where any wait on advancing federate time is placed. If no RTI-to-NetworkSim communications are necessary the simulation proceeds normally. In the event some communications are necessary, the advanceTime() function is a blocking call within which NetworkSim receives/sends interactions from/to the RTI. An internal dispatch mechanism routes these interactions to the appropriate protocol module which interprets them and can schedule new internal OMNeT++ messages. A similar mechanism interprets and routes OMNeT++ messages bound for external dispatch into the RTI.

4.2. Matlab/Simulink - UAV Plant and Controller Federate

Simulink [5] is a widely used simulation environment for dynamic and embedded systems such as communications, controls, and signal processing. It uses a set of pre-built block libraries for designing and controlling the simulation.

Integration of the Simulink simulation platform is similar to that of the OMNeT++ platform in that all of the platform-specific glue code is generated based on the overarching integration model. The model interpreter generates code that, in conjunction with several generic classes, is used to directly link any Simulink model with the C2 Wind Tunnel framework. The generic classes are completely reusable and take care of all basic RTI integration requirements: providing interfaces for converting between Matlab types and RTI types, encapsulating interfacing with the RTI for initializing the federate, setting the federate's lookahead value, and managing any publish and subscribe relationships with other federates. The automatically generated Java and Matlab files for the platform integration depend on this reusable code. Also, for each interaction that the Simulink model publishes or subscribes to, a corresponding .m file is generated. These serve as input and output bindings for communication via the RTI.

From the platform-specific modeling perspective, the user must add into the Simulink model an S-function block for each interaction to which the model either publishes or subscribes. It is via these blocks that the Simulink platform interacts with the remainder of the federation. The modeler must specify whether the block either publishes or subscribes with an interaction. This is done by setting the name of the .m file that the S-function will call. The modeler must also tell the S-function block which interaction it should call. This is done by passing the name of the interaction via a string parameter to the block. The naming convention of the .m files and of the parameters is standardized and easily derived from the primary GME model. Once the S-function blocks have been incorporated and their values set, no further manual steps are typically necessary for integration. Some effort may have to be spent to properly order the signals entering and existing the S-function blocks so that they correspond to the attribute ordering of the corresponding RTI interaction.

The key mechanism for synchronizing the clock progression of the Simulink model with that of the RTI is the basic time-progression model for S-function blocks. In Simulink each model is consulted when it can generate an
output -- this is done by calling the block method 
mdlGetTimeOfNextVarHit(). Here, the auto-
generated code places hooks to synchronize the 
model with the RTI and to allow logical time to 
progress only when the RTI allows the federate 
time to proceed. We keep the time-resolution 
low (~0.1 seconds) to minimize the error due to 
the synchronization of input and output events 
from the Simulink models. As for incoming 
interactions, the glue code uses a polling 
scheme at every time step to check if the 
federate has received an input.

Possible performance penalties due to having 
small time-resolution were evaluated. A smaller 
time-resolution is desirable in order to minimize 
simulation errors. After thorough evaluation, we 
found that the performance penalty is negligible 
in comparison to the basic lock-stepped 
simulation we use for synchronizing several 
time-constrained and time-regulating federates.

4.3. CPN - Organizational Decision 
Making Federate

Perhaps the greatest goal in command and 
control simulations is to evaluate the response 
of decision makers to the evolving situation. 
The C2 Wind Tunnel environment integrates 
Colored Petri Nets (CPN) to model and simulate 
human decision-making organizations.

We use CPNTools [8] augmented with the 
BRITNeY [9] extension. This extension provides 
access to lower level functionality, which was 
necessary for integration.

The primary challenge involved in integrating 
the CPNTools platform into the C2 environment 
was correct time synchronization. In order to 
sure the CPN model execution stops at 
desired times one extra place and transition, 
which is set to fire with a predefined frequency, 
were added into the model. The CPN platform 
optimistically progresses ahead of the HLA 
clock, but when needed, it can be rolled back to 
a desired time. This save and restore 
functionality might be useful for increasing 
performance using an optimistically large step 
size and lookahead. However, with our 
experiments, we found that the performance 
penalty incurred by using the small step size and 
lookahead was negligible compared to running 
several federates synchronously. Thus, we 
currently use a step size of ~1 second and 
lookahead of ~0.1 seconds for the CPN 
federate. Internally, while executing the CPN 
model via the BRITNeY Java library, the CPN 
clock moves forward 1 millisecond at a time. 
While time progresses internal to the CPN 
simulation, we compare its current time with the 
time granted by the RTI to the CPN federate. If 
the CPN cannot proceed in time, it requests the 
RTI to advance time and waits until it gets 
permission.

CPN models are imported into the GME 
modeling environment via a custom interpreter. 
Upon importing a CPN model, a CPNFederate 
element is created and the CPN places become 
corresponding ports on this federate. The ports 
must then be connected to either interactions or 
objects to specify inputs and outputs for the 
CPN model. These connections can represent 
either a publish or a subscribe relationship. This 
graphical step is all the integration effort 
necessary for CPN models. All code to 
communicate via the RTI and to synchronize the 
CPN federate with the rest of the federation is 
automatically generated from the GME 
integration model.

A custom GME model interpreter generates 
an XML file that describes all of the input-output 
bindings. The run-time CPN execution engine 
reads this file and simulates the CPN according 
to its specification. The set of places to monitor 
during execution can also be specified. Tokens 
on these monitored places are shown in a 
simple GUI component of CPNTools during run-
time.

5. Simulation Deployment 
and Execution

The deployment and execution of large-scale 
heterogeneous simulations can be quite 
complex. Typically, deployments span multiple 
computers and execution requires the 
coordination of many independent processes.

The HLA specification does not provide any 
mandate for how simulations are to be deployed 
or controlled, and available RTIs also do not 
provide any such facilities. As command and 
control scenarios grow larger they must span 
multiple computers and the deployment of multi-
domain simulations can impose a significant 
administrative burden. Manual approaches, 
such as hand-crafted batch files, do not scale 
well and are poorly suited to highly dynamic 
environments where deployment parameters 
change frequently. Our team has made 
additional contributions in this area in order to
ease the administrative burden imposed by these complex environments.

5.1. Deployment Modeling

The C2 Wind Tunnel team encountered numerous hurdles as we tried to deploy scenarios built upon our environment. As the complexity of our scenarios grew, manual processes soon became untenable. Our solution to this problem was to incorporate a model for deployment and execution directly into our central modeling environment. As shown in Fig. 6, the C2 Wind Tunnel meta model is augmented with several additional elements: Experiment, Host, Computer, Network, and Deployment. Now a single model incorporates both the federation integration design and the deployment information. With this extension, a model interpreter automatically generates all of the necessary scripts and files, copies the files to the appropriate computers, and prepares the environment for execution.

As discussed in previous sections, the overall simulation model is a composition of federates and their relationships via interactions and objects. For any given experiment, a simulation scenario may only utilize a subset of the federates defined in the model. Similarly, each platform-specific model involved may be parameterized to allow for run-time flexibility. Example parameters are the duration of a network attack (for network simulator engine) or the weight a given command decision may be given (for a CPNTools engine). An experiment is the set of federates included in a specific deployment and their run-time parameterization.

Frequently an experiment is run on more than one hardware setup. A designer may run the entire simulation on one machine during development, while deploying the simulation onto a cluster for full-scale demonstrations. The network element of the above meta model is the physical set of computers involved in a specific deployment.

The deployment element is where an experiment configuration is mapped to a network configuration. Specific federates are assigned to hosts in the network. Thus, allowing complete flexibility in defining which simulation tools execute on which hardware.

A custom interpreter reads the deployment configuration from the model and generates all of the script files necessary to support the deployment. In cases where modeling deployments may only be partially specified, for example in very large-scale or rapidly changing environments, the interpreter generates the deployment for whatever portion is defined. Once generated, the environment is fully prepared for experiment execution.

Finally, the generated scripts manage the actual movement of files and code to the various hosts being used. Upon invocation, the scripts remotely connect to each machine and create local copies of all necessary files before the simulation begins execution. The scripts then coordinate the execution of all federates. After
the experiment is concluded, the scripts remotely stop all processes, collect output files, and clean all local copies to restore the hardware to its original state.

5.2. Federation Manager

The HLA standard does prescribe some methods for controlling the execution of a simulation. However, the C2 Wind Tunnel environment extends much greater control and coordination of federate execution throughout a simulation. This is achieved by using a special federate called the Federation Manager (FM).

The FM is a generic federate, and so can be used as a part of any federation. It coordinates a simulation by: (1) waiting for all federates in an experiment to join the federation before allowing it to begin simulation, and (2) making sure all federates are initialized and ready to begin the simulation before allowing it to proceed.

The first item above is achieved by listing which federates are part of the simulation in a configuration file that is read by the FM upon its initial execution. Using this information, along with the HLA’s built-in “FederateObject” object-class, the FM can detect when each federate joins the federation, and allow the simulation to proceed only when all federates have joined.

Making sure all federates are fully initialized is necessary to avoid one or more federates proceeding with simulation execution before others are ready. Such behavior would corrupt and therefore invalidate the simulation. The FM uses “synchronization points,” as specified by the HLA standard, to guarantee that all federates are ready to proceed with the simulation before any of them to proceed. In particular, it registers a synchronization point to allow federates to report when they are initialized and ready to proceed with the simulation. Once all federates have reported that they have reached the synchronization point, the FM allows the simulation to proceed.

The coordination the FM exerts over a simulation is extremely important in that it allows simulations to be easily repeated. Without the FM, it would be nearly impossible to guarantee, for any sufficiently complex simulation, that all federates involved are running and start the simulation simultaneously.

The FM also allows the user to exercise greater control over the simulation. This is realized via several different mechanisms:

First, the FM is capable of by pacing the simulation in synchronization with the wall-clock, or allowing the simulation to run as fast as possible. This is accomplished by coding the FM to monitor the wall-clock and to use RTI calls to keep the wall-clock and the simulation clock synchronized. This behavior can be turned on and off using the FM’s GUI. Turned off, the FM allows the simulation to proceed as fast as hardware and network speeds allow.

Second, the FM provides fine-grained controls to allow for the simulation to paused for examination (and subsequently resumed), or terminated, at any point in the simulation. This works by placing “Pause,” “Resume,” and “End” interactions directly into the federation model (see Fig. 1). The FM sends out “Pause”, “Resume” and “End” interactions either on-demand (via GUI buttons) or at times pre-specified in its configuration file. Each federate is coded to respond to these interactions automatically by way of the automatically generated glue-code.

Third, the FM also allows federation specific interactions to be injected into the simulation at pre-specified times. This is very useful for both debugging and quick what-if considerations. This functionality is controlled by specifying in the FM configuration file which interactions are to be injected with what parameter-values and at what times.

Finally, the Federation Manager allows interactions to be monitored and logged as they are sent by federates during a simulation. This is also specified in the FM configuration file. Monitored interactions, as they occur, are displayed in a text box in the FM’s GUI.

The deployment of scenarios and the management of their execution is typically an error-prone and time consuming administrative task. The C2 Wind Tunnel greatly reduces this overhead by directly incorporating the deployment information into the central modeling environment, by automating all of the execution and cleanup steps and by providing comprehensive tools to manage the environment during run-time execution.

6. Experimentation with the Platform

The C2 Wind Tunnel project’s goal is to create a heterogeneous simulation environment tailored towards scenarios involving, for instance, the interaction of multiple unmanned aerial vehicles, their operating conditions, and the associated
command and control organization and infrastructure. Rapid evolution of scenario details and easy evaluation of command and control effectiveness are key motivators. A typical scenario involves the deployment of one or more UAVs (implemented using Simulink models) into a combat zone. The deployment zone and the physical positioning of the ground and aerial vehicles are modeled using a custom Java federate and visualized using Google Earth. The UAVs may have objectives including: data collection, target acquisition and engagement, or battle damage assessment. Video sensors (implemented and simulated using a Java federate) mounted on the UAVs must collect information and relay it via a communications network (OMNeT++) back to a centralized decision making organization (a CPN model). The organization must react appropriately to the information and provide guidance to the vehicles. In addition, the UAVs are themselves highly autonomous and must utilize collected sensor data to pursue their given objectives.

Our first goal was to demonstrate that we could rapidly synthesize such simulations using our model-based integration environment involving all the domain-specific simulation platforms described above. For demonstration purposes, we used an urban scenario with four blue UAVs and two red ground vehicles. All UAVs have video sensors and are continuously transmitting video data to the control station. The control station remotely controls the UAVs in a formation flight and assesses the targets one by one based on the initial position estimates of the targets.

All network communications are simulated using an OMNeT++ federate. We tested how a network attack affects mission performance. The abstraction of low-level physical layer communication in OMNeT++ makes it straightforward to implement network attacks. Attacks that we found most reasonable in our context are Distributed Denial of Service (DDOS) attacks.

Using a collection of “zombie” nodes in dedicated sub-networks, either parameterized or controlled by a master node, any type of “dumb” or non-adaptive attacks can be simulated. For our purposes, DDOS was sufficiently disruptive. Each zombie machine sends, at specified intervals, service requests to every discovered valid node. As communications degrade, the effect on formation flight is pronounced. The formation flight does not loosen; rather it collapses altogether. The UAVs continue in their individual directions entirely. Sensor information is lost, so the operator becomes unable to repair the loss manually.

The impact of these attacks on the command and control of the UAVs closely mirrors both the theoretical and real-world consequences observed previously in other contexts. This gives the experimenters confidence that the results of simulation can be directly applied to the modeled scenarios.

Table 1 captures the platform type and update rate for each of the federates involved in our simulation.

<table>
<thead>
<tr>
<th>Federate</th>
<th>Platform</th>
<th>Update Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>UAV Dynamics</td>
<td>Simulink</td>
<td>100hz</td>
</tr>
<tr>
<td>Physics</td>
<td>Java</td>
<td>10hz</td>
</tr>
<tr>
<td>Visualization</td>
<td>G. Earth</td>
<td>10hz</td>
</tr>
<tr>
<td>Comm. Network</td>
<td>OMNeT++</td>
<td>20hz</td>
</tr>
<tr>
<td>Decision Making</td>
<td>CPNTools</td>
<td>1hz</td>
</tr>
</tbody>
</table>

Table 1: Model Information for Experimental Scenario

Our hardware configuration for demonstrations consists of six dual-core 3.0GHz-class machines networked via a dedicated 100Mb/s switch all with nVidia GTX 280 graphics cards. In a typical deployment, each machine had one to three federates running on it. Despite highly complex platform specific models we have not experienced any significant performance bottlenecks during simulations lasting an average of thirty minutes. If our scenarios are deployed entirely onto one of our typical development machines (a clone of those in our demonstration cluster) performance is acceptable but noticeably slower. This is especially true if multiple visualization federates are present.

7. Conclusion and Future Work

Integration of complex command and control simulations composed of numerous heterogeneous platforms is a challenging problem. Each simulation platform may have its own operational semantics and requires integration at not only the platform level but also at the platform-specific model level.

Pervasive use of models throughout simulation platforms opens the door to the use of model-integrated methodologies for defining
integration among these tools. In this paper we discussed the use of the HLA and GME tools to create a comprehensive modeling environment for heterogeneous simulation. In this environment it is possible to rapidly integrate domain-specific models from diverse simulation platforms and to dynamically generate all of the needed configuration and integration code. The environment also provides automated facilities to manage the deployment and execution of the simulation itself. Together these tools greatly reduce the time required to design, modify, and test command and control scenarios.

In the future, we expect to integrate additional simulation platforms to expand the range of possible scenarios. Consequently, new simulation platforms will require expanding the existing integration meta model. These enhancements should allow for greater scenario flexibility and reduced development, configuration, and operational costs. Additionally, we are exploring alternatives to include capabilities in the C2W framework to alter and configure the entire simulation during run-time.

8. Acknowledgements

The authors acknowledge financial support from the US Air Force Office of Scientific Research under the project “Human Centric Design Environments for Command and Control Systems: The C2 Wind Tunnel.” We also acknowledge the invaluable contributions to this project from our collaborators at George Mason University, University of California at Berkeley, and Arizona State University.

9. References

[16] The authors acknowledge financial support from the US Air Force Office of Scientific Research under the project “Human Centric Design Environments for Command and Control Systems: The C2 Wind Tunnel.” We also acknowledge the invaluable contributions to this project from our collaborators at George Mason University, University of California at Berkeley, and Arizona State University.

9. References
