ABSTRACT

This paper presents the Networked Control Systems Windtunnel (NCSWT), an integrated modeling and simulation tool for network control systems (NCS). The tool integrates Matlab/Simulink and ns-2 to model and simulate NCS. The tool integrates time-triggered execution semantics implemented in Matlab/Simulink and discrete event execution semantics in ns-2 using the High Level Architecture (HLA). Our implementation of the NCSWT tool based on HLA guarantees accurate time synchronization and data communication in heterogenous simulations. NCSWT uses the model integrated computing (MIC) techniques to define HLA-based model constructs such as federates representing the simulators and interactions representing the communication between the simulators. NCSWT also uses MIC techniques to define models representing the control system dynamics and networking system behaviors of NCS for the rapid synthesis of simulations.

Categories and Subject Descriptors
I.6.7 [SIMULATION AND MODELING]: Simulation Support Systems—Environments

General Terms
Design, Experimentation

Keywords
Modeling, Simulation, Networked Control Systems, HLA

1. INTRODUCTION

Networked control systems (NCS) have gained increasing attention in recent years due to their cost effective and flexible applications [5]. NCS are often employed in critical settings, hence the assurance of properties such as performance, stability, safety and security are very essential. Currently, many NCS are designed without considering the effects of the network operating environment (e.g. time-varying delays and packet losses). Such inadequacies in the system design phase can lead to catastrophic consequences when the actual systems are deployed, as these systems are interconnected to open networks and become exposed to network dynamics and uncertainties.

As NCS become increasingly complex through distributed architectures and expanded mission capability, it becomes more challenging to formally analyze their performance, stability, safety and security properties. As a result, there is a pressing need to evaluate both the control system and the networking system together for a rapidly growing number of applications, such as unmanned aerial vehicles (UAVs) and industrial control systems. Simulation is a powerful technique for evaluation and can be used at various design stages. Without the support of appropriate tools the simulation process can be extremely time-consuming and error-prone, if at all possible.

Currently, most of the existing individual simulators have limited capability in simulating NCS. For example, Matlab/Simulink is a very popular tool to model and evaluate the performance of control systems [4]. Although network simulation is provided in Matlab/Simulink using add-ons such as TrueTime [7], the accuracy of the simulation depends on the level of abstraction of the network protocol models. For example, the network protocol in TrueTime only supports link layer protocols but does not support higher level protocols such as TCP or UDP protocols, which are essential. Packet-level network simulators such as ns-2 [8], provide a detailed implementation of the network stack. Yet using ns-2 alone for NCS evaluation requires the control algorithm to be fully implemented in a high-level language such as C++. This becomes very difficult as the complexity of the modeled system increases.

In order to develop a realistic and accurate simulation of NCS, we need a modeling and simulation environment that can integrate heterogeneous tools. The integration, although very beneficial, introduces additional challenges. The first challenge is the design-time scalability of modeling NCS which involves the ability to rapidly design and model NCS of various complexity and size. The second challenge is time synchronization of the heterogenous simulation. Given that the simulators have different execution semantics, the time synchronization involves the progression of time in the heterogenous simulation in order to ensure the accurate sequence of events in realizing the simulation. The third challenge involves the data communication between the heterogenous simulators to ensure that data is passed between the simulators in an appropriate manner to ensure the accuracy of the simulation. Finally, the fourth challenge involves the run-time scalability which involves the ability of the simulation environment to handle the simulation of large and complex NCS.

Several efforts have been made towards integrating multiple sim-
ulators in order to effectively simulate NCS. A tool chain PiccSIM was developed in [13] that allows the integration of Matlab/Simulink models with ns-2. PiccSIM also provides a graphical user interface for the design of networked control systems and the automatic code generation of ns-2 and Matlab/Simulink models. In [14], a special simulator coupling, implemented in C/C++ is used to integrate the simulators, ModelSim, Matlab/Simulink and ns-2 to establish the communication between the simulators. Other tool integration projects also targeted for networked control systems include [15] and references therein. In contrast to all the other tools, our approach provides a model-based approach that tightly integrates the design of the control and network layers in NCS providing a clear modeling of the information exchange between the two layers. Also, our integration approach uses a standard based on the high level architecture (HLA) for the integration of heterogeneous simulators. In [16], we presented a preliminary version of our integrated tool for simulating NCS based on the HLA framework. The tool in [16] lacks design-time scalability and this makes it very tedious and error prone to model and simulate NCS of varying complexity.

In order to address all the challenges identified with developing an accurate simulation environment for NCS, we present a heterogeneous simulation tool for NCS, called the networked control systems wind tunnel (NCSWT) [17], which combines the network simulation capabilities of ns-2 with the control design capabilities of Matlab/Simulink. NCSWT addresses the challenge of design-time scalability which involves the ability to rapidly model and simulate NCS of various complexity under various scenarios, by adopting the model integrated computing (MIC) techniques [18]. MIC is an approach to the development of complex software systems, applicable in all phases - analysis, design, implementation, testing, maintenance and evolution. Based on MIC approach, we present three domain specific modeling languages (DSMLs), the NCWT model integration language, the Control Layer Modeling Language and the Network Layer Modeling Language. The DSMLs are developed using the generic modeling environment (GME) [19].

NCSWT addresses the challenges involving time synchronisation and data communication by adopting the high level architecture (HLA) framework for the implementation of the simulation environment [20]. HLA is a standard for simulation interoperability developed to allow many independently developed simulations, each designed for a particular problem domain, to be combined into a larger and more complex simulation. In HLA, the independent simulators are known as federates and the larger simulation formed by the interconnection of the federates is known as the federation. HLA standard provides a set of services to accurately handle time management and data distribution among the heterogeneous simulators.

Finally, we demonstrate the NCSWT tool through an evaluation as will be described in Section 4. We list the required software packages for NCSWT tool and provide the design-time efficiency and run-time efficiency for a specific NCS case study.

2. NCSWT

Figure 1 shows an overview of the NCSWT tool architecture. The architecture is composed of two main parts, the design-time models and the run-time components.

2.1 Design-Time Models

In Figure 1, the design-time models are used to define the NCS and its components using representative models in order to enable the realization of a HLA-based simulation of the NCS. The design-time models are defined by three domain specific modeling languages (DSMLs).

2.1.1 NCSWT Model Integration Language

The NCSWT Model Integration Language (NCSWT MIL) provides the modeling primitives to specify the NCS in terms of HLA-based constructs, such as federates representing the simulators and interactions representing the communication between the simulators, in order to execute a HLA-based simulation. An instance model created using NCSWT MIL is referred to as the base architecture of the NCS. The base architecture describes each component of the NCS in terms of federates representing the simulator for the respective component and interactions representing the communication between the federates. From the base architecture model, the executables for configuring the run-time environment for a specific NCS is generated. The NCSWT MIL is an extension of the work in [21] which introduced the C2WT, a DSML for HLA-based simulations. The NCSWT MIL describes the tight coupling between the control and network layers of the NCS by defining how the two layers interact. Figure 1 shows the NCSWT MIL meta-model. The federates NCSWTFederate and NS2Federate shown in the NCSWT MIL meta-model represent the modeling concepts for the Matlab and ns-2 federates describing the respective simulators in HLA.

The interactions NetworkInteraction, CrossLayerInteraction and ApplicationInteraction define the communication between the federates. The NetworkInteraction enables the modeling of packets exchanged over the communication network. The CrossLayerInteraction allows for the modeling of information exchange between the network and application layers of a network protocol stack while the ApplicationInteraction allows for the modeling of information exchanged between components of the control system that are transmitted or received by other means other than a communication network.

2.1.2 Control Layer Modeling Language (CLML)

The Control Layer Modeling Language (CLML) defines the mod-

Figure 1: Overview of NCSWT

In Figure 1, the design-time models are used to define the NCS and its components using representative models in order to enable the realization of a HLA-based simulation of the NCS. The design-time models are defined by three domain specific modeling languages (DSMLs).
The RTI, an implementation of the HLA standard, manages the communication between federates. A model created by CLML is a refinement of the base architecture model of a NCS, created in the NCSWT MIL, with the details regarding the dynamics of the control system added. In order to maintain consistency with the base architecture model defined in the NCSWT MIL, a model transformation is used to transform the base architecture model to a base model in CLML. Then the control design concepts in CLML are used to define the dynamics of the components in the NCS. In CLML, using a set of modeling primitives such as $T_s$ (Sampling Time), ModelName and ModelLibraryName etc., a user can specify the model and parameters that define the dynamic behavior of a control system component. Using the defined parameters in CLML, run-time Matlab/Simulink code can be generated for implementing the control system.

2.1.3 Network Layer Modeling Language

The Network Layer Modeling Language (NLML) defines the modeling primitives for defining the dynamics of the communication network. This includes the capacity, loss models, routing and other additional properties to realize a desired networked control system. Similar to CLML, a model transformation is used to transform the base architecture model to a base model in NLML. Then the network primitives defined in NLML are used to define the network properties for the NCS. A user can then specify the transport agent, loss model of network links and other various network properties to simulate the network dynamics. Run-time network configuration and model scripts can then be generated based on the defined model parameters for deployment on ns-2.

2.2 Run-Time Components

In Figure 2 the run-time components represent the main software components and interfaces for the actual realization of a simulation using the HLA framework. These components include the Run-Time Infrastructure (RTI), the federates and all the necessary glue code for the interfaces as well as monitoring tools for visualizing and evaluating the results.

2.2.1 Run-Time Infrastructure (RTI) and Federates

The RTI, an implementation of the HLA standard, manages the communication between different federates. Using interactions, federates communicate between each other through the RTI. The RTI handles the coordination of time and data passed between federates. A number of commercial and academic RTI implementations are available. Currently, we use Portico version 1.0.2, an open source cross-platform HLA implementation which supports both C++ and Java clients.

Each federate has a single point of contact to the RTI through which it can communicate with other federates. Each federate represents a single instance of the corresponding simulator' interface to the RTI. For example, the NS-2 federate is a software component that interfaces the ns-2 simulator with the RTI.

We briefly describe the main services provided by the RTI.

(a) Time Synchronization: In a HLA-based federation, each federate has its own logical time. The RTI preserves the causality of the federation by ensuring that no simulation receives an event that occurred in the past relative to its own. The RTI ensures the accurate progression of time through the use of a time advance grant request (TAR) and time advance grant (TAG) mechanism [11]. For ns-2, this mechanism is integrated into the ns-2 scheduler while in Matlab/Simulink, the mechanism is integrated as part of the interface code to the Matlab federate.

(b) Data Communication: The RTI uses a publish-and-subscribe mechanism for passing messages and object updates through the federation in order to ensure the accurate data communication and coordination between the federates [12]. The type of messages exchanged between the federates are defined by the interactions modeled in the NCSWT MIL during the design-time modeling and is integrated in the generated code deployed during the run-time simulations.

In addition to these two main services, the RTI provides additional services for monitoring the results of NCS simulations. Also, the simulators ns-2 and Matlab/Simulink are each equipped with visualization and analysis tools to enable the monitoring and evaluation of results from NCS simulations.

3. EVALUATION

The simulation of a NCS using the NCSWT tool requires two major steps. The first step involves the modeling of a NCS and generation of all the necessary models and glue code for the simulation of the NCS. The modeling of the NCS is performed in GME using the three DSMLs discussed in Section 2.1. The modeling and code generation is performed on a computer running a Windows operating system. The second step involves the deployment of the generated code and models and the execution of the simulation. The simulation is executed on a computer running a Linux operating system.

The NCSWT tool requires the software packages as shown in Table 1. Matlab/Simulink and ns-2 packages are used for the simulation of the control layer and network layer of the NCS respectively. Portico 1.0.2 is the RTI implementation of the HLA used for running the federation. GME is the graphical environment used for the modeling and generation of all the necessary components for the simulation of the NCS. Microsoft Visual Studio is used for the execution of the code generators and model transformations from the three DSMLs. Eclipse is used for the compilation of the run-time components required for the simulation.

We have evaluated the NCSWT tool using various realistic NCS case studies. The case studies include an NCS composed of a single plant and controller [13], NCS composed of multiple plant and multiple controllers [14] and also the rate adaptation in a multi-agent NCS with bandwidth constraints which shows how the dynamics in the network affects the sampling rate of the control systems [15]. NCSWT provides flexibility with running simulations especially with testing various configurations for a specific NCS case study. For example, in order to test various network configurations for a particular NCS, we only need to modify the configurations in the

![Figure 2: NCSWT Meta Model](image-url)
Table 1: Required Software Packages

<table>
<thead>
<tr>
<th>Files</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Matlab/Simulink</td>
<td>204 Kilobytes</td>
</tr>
<tr>
<td>2. ns-2</td>
<td>132 Kilobytes</td>
</tr>
<tr>
<td>3. Portico 1.0.2</td>
<td>100 Kilobytes</td>
</tr>
<tr>
<td>4. Generic Modeling Environment (GME),</td>
<td>160 Kilobytes</td>
</tr>
<tr>
<td><a href="http://www.isi.vanderbilt.edu/projects/gme">www.isi.vanderbilt.edu/projects/gme</a></td>
<td></td>
</tr>
<tr>
<td>5. Microsoft Visual Studio 2008 or later,</td>
<td>4 Kilobytes</td>
</tr>
<tr>
<td><a href="http://www.microsoft.com/vissimstudio">www.microsoft.com/vissimstudio</a></td>
<td></td>
</tr>
<tr>
<td>6. Eclipse</td>
<td>22 Kilobytes</td>
</tr>
<tr>
<td><a href="http://www.eclipse.org">www.eclipse.org</a></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Generated Code for NCS Example

<table>
<thead>
<tr>
<th>Files</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Matlab models</td>
<td>100 Kilobytes</td>
</tr>
<tr>
<td>2. Matlab glue code</td>
<td>132 Kilobytes</td>
</tr>
<tr>
<td>3. ns-2 model and topology scripts</td>
<td>20 Kilobytes</td>
</tr>
<tr>
<td>4. ns-2 glue code</td>
<td>160 Kilobytes</td>
</tr>
<tr>
<td>5. Federation startup script</td>
<td>4 Kilobytes</td>
</tr>
</tbody>
</table>

Table 3: Time Efficiency for NCS Example

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Actual Duration (in minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal</td>
<td>5.5</td>
</tr>
<tr>
<td>Packet Losses</td>
<td></td>
</tr>
<tr>
<td>20%</td>
<td>8.4</td>
</tr>
<tr>
<td>30%</td>
<td>11.4</td>
</tr>
<tr>
<td>40%</td>
<td>13.5</td>
</tr>
<tr>
<td>Multi-hop Network</td>
<td></td>
</tr>
<tr>
<td>3 hops</td>
<td>17.4</td>
</tr>
<tr>
<td>4 hops</td>
<td>22.2</td>
</tr>
<tr>
<td>5 hops</td>
<td>26.2</td>
</tr>
</tbody>
</table>

4. CONCLUSION

We presented the integration tool, NCSWT for the modeling and simulation of networked control systems. We described the HLA-based approach guiding the tool’s implementation as well as the MIC techniques for the rapid synthesis of components required for the simulation of a NCS. Additionally, we provided an evaluation of tool.

5. REFERENCES