Simulation Modelling Practice and Theory 27 (2012) 90–111

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

NCSWT: An integrated modeling and simulation tool for networked control systems

Emeka Eyisi [a,*], Jia Bai [a], Derek Riley [b], Jiannian Weng [a], Wei Yan [a], Yuan Xue [a], Xenofon Koutsoukos [a], Janos Sztipanovits [a]

[a] Institute for Software Integrated Systems, Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, USA
[b] Department of Computer Science, University of Wisconsin-Parkside, Kenosha, WI, USA

ARTICLE INFO

Article history:
Received 21 December 2011
Received in revised form 7 April 2012
Accepted 3 May 2012
Available online 20 June 2012

Keywords:
Modeling
Simulation
Networked control systems
HLA

ABSTRACT

Networked Control Systems (NCS) are becoming increasingly ubiquitous in a growing number of applications, such as groups of unmanned aerial vehicles and industrial control systems. The evaluation of NCS properties such as stability and performance is very important given that these systems are typically deployed in critical settings. This paper presents the Networked Control Systems Wind Tunnel (NCSWT), an integrated modeling and simulation tool for the evaluation of Networked Control Systems (NCS). NCSWT integrates Matlab/Simulink and ns-2 for modeling and simulation of NCS using the High Level Architecture (HLA) standard. The tool is composed of two parts, the design-time models and the run-time components. The design-time models use Model Integrated Computing (MIC) to define HLA-based model constructs such as federates representing the simulators and interactions representing the communication between the simulators. MIC techniques facilitate the modeling and design of complex systems by using abstractions defined in domain-specific modeling languages (DSMLs) to describe the systems. The design-time models represent the control system dynamics and networking system behaviors in order to facilitate the run-time simulation of a NCS. The run-time components represent the main software components and interfaces for the actual realization of a NCS simulation using the HLA framework. Our implementation of the NCSWT based on HLA guarantees accurate time synchronization and data communication. Two case studies are presented to demonstrate the capabilities of the tool as well as evaluate the impact of network effects on NCS.

1. Introduction

Networked Control Systems (NCS) have gained increasing attention in recent years due to their cost effective and flexible applications [1–4]. NCS are comprised of plants (the systems or processes to be controlled), actuators, sensors, and controllers, that exchange information (reference input, plant output, control input, etc.) over a communication network. NCS are often employed in critical settings, therefore the assurance of properties such as stability, performance, safety and security are essential. Currently, many NCS are designed based on simplifying assumptions on the network, specifically in regards to their network operating environment (e.g. time-varying delays and packet losses). These assumptions, although they make

* Corresponding author. Tel.: +1 732 371 8048.
E-mail addresses: emeka.p.eyisi@vanderbilt.edu (E. Eyisi), jia.bai@vanderbilt.edu (J. Bai), rileyd@uwp.edu (D. Riley), jiannian.weng@vanderbilt.edu (J. Weng), wei.yan@vanderbilt.edu (W. Yan), yuan.xue@vanderbilt.edu (Y. Xue), xenofon.koutsoukos@vanderbilt.edu (X. Koutsoukos), janos.sztipanovits@vanderbilt.edu (J. Sztipanovits).

1569-190X/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2012.05.004

the analysis of NCS tractable, sometimes do not fully represent the real network dynamics. Such limitations in the system design phase can lead to catastrophic consequences when the actual systems are deployed, as the overall system behavior depends on network dynamics and uncertainties.

Numerous techniques aim to formally analyze NCS properties such as stability, performance, safety, and security [3,5,6]. In addition, various control approaches have been developed to tolerate and compensate for the impact of various network uncertainties in NCS. Some of these approaches include the development of new control paradigms to improve performance and security of NCS such as in [7], passivity-based methods for achieving robustness to network uncertainties [8], gain-scheduling techniques [9], model-predictive techniques [10], deadband techniques for efficient network resource usage [11] and NCS model-based techniques [12,13]. These approaches and the references therein attempt to address different challenges but as NCS become increasingly complex, it becomes more challenging to formally analyze these NCS 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, but it requires the support of appropriate tools during the design-time and run-time stages in order for the process to be efficient and less prone to errors.

Currently, several simulators have been used for simulating NCS but have limited capabilities. For example, Matlab/Simulink is a very popular tool to model and evaluate the performance of control systems [14]. Although network simulation is provided in Matlab/Simulink using add-ons such as TrueTime [15], the accuracy of the simulation depends on the level of abstraction of the network protocol models. Specifically, the network protocol in TrueTime only supports link layer protocols but not higher level protocols such as TCP or UDP protocols, which are essential for simulating the communication network of a NCS. Packet-level network simulators such as ns-2 [16], provide a detailed implementation of the network stack for packet level data transmission. Yet, using ns-2 only 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 NCS increases. In order to develop a realistic and accurate simulation of NCS, we need a modeling and simulation environment that can integrate existing tools for the accurate simulation of the control dynamics as well as the networking system of a NCS.

The integration of existing tools for the accurate simulation of NCS, although very beneficial, faces several challenges. The first challenge is the design-time scalability of modeling NCS. This involves the ability to rapidly design and model NCS of various complexity and size. The second challenge is time synchronization of the heterogeneous simulation components during execution. Given that the simulators operate in potentially different time scales using disparate time models, time synchronization between the simulators is critical to preserve the correctness of the simulation. The third challenge involves the data communication between the simulators to ensure consistent data semantics during the simulation. Finally, the fourth challenge involves the run-time scalability which is the ability of the simulation environment to handle the simulation of large and complex NCS.

In order to address these challenges, we present an integrated modeling and simulation tool for NCS, called the Networked Control Systems Wind Tunnel (NCSWT),[1] which combines the network simulation capabilities of ns-2 with the control design and simulation capabilities of Matlab/Simulink. NCSWT addresses the challenge of design-time scalability of modeling NCS described previously by adopting Model Integrated Computing (MIC) [17]. MIC is an approach for the development of complex software systems, applicable in all phases – analysis, design, implementation, testing, maintenance and evolution. The key idea in MIC is to create domain-specific modeling languages (DSMLs) using a meta-modeling framework and then describe objects in terms of the domain-specific models. These models are formally represented and can be checked for correctness against pre-specified design rules and can be programmatically traversed and transformed to produce/or modify code and other engineering artifacts. Often, these models are transformed into alternate but equivalent representations, which can be used by external analysis and simulation tools to verify certain properties of the system [17]. We present three domain specific modeling languages (DSMLs), the NCSWT model integration language (NCSWT MIL), the Control Design Modeling Language (CDML) and the Network Design Modeling Language (NDML). The DSMLs, developed using the generic modeling environment (GME) [18], facilitate the rapid design and modeling of NCS. In addition, the DSMLs and the NCSWT framework are designed to ensure the consistency of data semantics among the simulators used in the simulation of a NCS.

NCSWT addresses the challenges involving time synchronization and data communication by adopting the High Level Architecture (HLA) for the implementation of the simulation environment framework [19]. HLA is a standard for simulation interoperability that allows 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. The HLA standard provides a set of services to accurately handle time management and data distribution among the heterogeneous simulators. In our framework, we utilize the time management services provided by the HLA to ensure that the time model in the control system simulated in Matlab/Simulink and the time model in the networking system simulated in ns-2 are synchronized. We also utilize the data distribution services to ensure the correct exchange of data between the simulations of the control dynamics and communication network of a NCS.

Finally, we provide case studies to evaluate the tool. The first case study involves an unmanned aerial vehicle which is remotely controlled over an 802.11b wireless network to follow a given reference trajectory. This case study is used to

[1] The NCSWT software package is available online at http://vanets.vuse.vanderbilt.edu/dokuwiki/doku.php?id=research:cps.

Table 1
Acronyms.

NCSs	Networked Control Systems
HLA	High Level Architecture
MIC	Model Integrated Computing
DSML	Domain Specific Modeling language
NCSWT	Networked Control Systems Wind Tunnel
MIL	Model Integration Language
CDML	Control Design Modeling language
NDML	Network Design Modeling language
TAG	Time Advance Grant
TAR	Time Advance Request
NER	Next Event Request

demonstrate the use of the NCSWT to evaluate the impact of network effects, such as packet loss, time-varying delays and multi-hop topologies, on a safety critical system. The second NCS case study is an industrial control system involving a network of three plant and controller pairs sharing a wireless communication channel and operating at different sampling rates. This case study is used to demonstrate the convenience and scalability of NCSWT tool in handling NCS of different complexities in time and size.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 provides an overview of the NCSWT. Section 4 presents the model-based design and integration of NCS using the NCSWT tool. Section 5 describes the NCSWT run-time components. Section 6 presents the implementation overview of the NCSWT tool. Section 7 presents the cases studies. Section 8 provides an evaluation of the NCSWT and finally Section 9 concludes the work. For ease of readability, a list of the main acronyms used throughout the paper is provided in Table 1.

2. Related work

There are two main approaches in using existing tools for the simulation of NCS. The first approach involves extending the features of an independent simulator to enable the simulation of both the control dynamics and communication network of a NCS. The second approach involves the integration of independent tools whereby each tool is designed for simulating a specific domain (control dynamics or the communication network) of a NCS.

The approach involving the extension of independent simulators for NCS simulation can be further categorized into approaches involving the extension of network simulators and approaches involving the extension of simulators for dynamical systems. In the approach involving the extension of independent network simulators, in addition to ns-2, there exist other network simulators such as OMNet++ [20] that have been used to simulated NCS, but these attempts have been met with similar challenges as attempts to extend the ns-2 simulator in regards to the difficulty of accurately modeling and simulating dynamical systems. Similarly, in addition to Matlab/Simulink, there exist other tools such as Modelica [21] and Ptolemy [22], which allow the construction of continuous and discrete dynamical systems. The challenge faced by these tools is the ability to precisely model the network stack in order to simulate the communication network of a NCS.

On the other hand, several efforts have been made toward integrating multiple simulators in order to effectively simulate NCS. PiccSIM presented in [23] allows the integration of Matlab/Simulink models with ns-2 and provides a graphical user interface for the design of NCS and the automatic code generation of ns-2 and Matlab/Simulink models. In [24], a special simulator interface, implemented in C/C++ is used to integrate the simulators, ModelSim, Matlab/Simulink and ns-2 to establish the communication between the simulators. Other integration tools for NCS include [25–27]. In contrast to these tools, our approach provides a model-based framework that tightly integrates the design of the control system and communication network in NCS providing a well-defined abstraction of the information exchange between the two design views. Also, our integration approach uses a standard based on the HLA for the integration of heterogeneous simulators which assures that the integration of our tools enforces the HLA time synchronization and data communication standards. In [28], we presented a preliminary version of the tool, which only provides a run-time implementation for simulating NCS based on the HLA framework. This paper extends the work in [28] by providing a model-based framework to address design-time scalability challenges in order to facilitate the efficient modeling and simulation of NCS based on the HLA framework. In addition, we present a new run-time implementation requiring only the Linux platform, compared to the previous run-time implementation, which required both Linux and Windows platforms [28]. The new run-time implementation eliminates the need of using socket communication.

3. Overview of the NCSWT

An overview of the NCSWT architecture is shown in Fig. 1. The architecture is composed of two main parts, the design-time models and the run-time components. The design-time models are used to define the NCS and its components 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 NCSWT Model Integration Language (NCSWT MIL), the Control Design Modeling Language (CDML) and the Network Design Modeling Language (NDML). These DSMLs, developed using the MIC approach [17], use

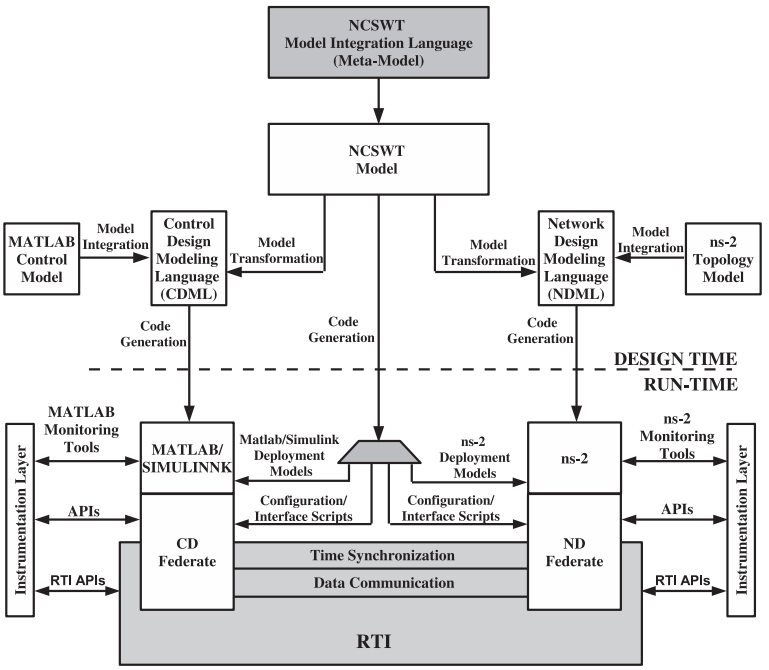


Fig. 1. Overview of NCSWT.

abstractions to define the NCS in specific modeling domains to enable the rapid modeling and design of NCS for simulation. The design-time models constructed from these DSMLs generate software components, interface glue code and configuration files for the NCS which is deployed and executed at run-time. The run-time components represent the main software components and interfaces for the actual simulation of the NCS using the HLA framework. These components include the Run-Time Infrastructure (RTI), the federates, the simulators (Matlab/Simulink and ns-2) and all the necessary configuration and interface scripts generated from the design-time models. These components also include the monitoring tools for visualizing and evaluating the results.

A detailed description of the design-time modeling framework and the run-time execution framework are presented in Sections 4 and 5 respectively.

4. Model-based design and integration

In order to facilitate the rapid generation and simulation of NCS with minimal effort, we employ Model Integrated Computing (MIC) techniques [17]. We define meta-models for three domain-specific modeling languages (DSMLs) for modeling and integrating the NCS in the HLA framework. The DSMLs are:

1. NCSWT Modeling Integration Language (NCSWT MIL)
2. Control Design Modeling Language (CDML)
3. Network Design Modeling Language (NDML)

NCSWT MIL is an extension of the base HLA meta-language [29]. Before we provide a description of each of the DSMLs, we provide a background of the base HLA meta-language.

4.1. Base HLA meta-language

The base HLA meta-language is a DSML which provides a graphical environment for designing and deploying heterogeneous simulations using model-based design techniques [29]. This DSML provides all of the modeling primitives required to specify the integration, deployment, and execution of a federated simulation. Once the integration model has been defined for a given environment, a set of reusable model interpreters are executed to automatically generate engine-specific glue code and all deployment and execution artifacts. All generation and deployment steps directly rely upon the initial integration model. Fig. 2 shows the primary portion of the base HLA meta-language DSML, specified using GME, that defines the composition elements. The three primary elements in a federation (defined by the model FOMSheet) are Interaction (on right-hand side of Fig. 2), Object (on the left-hand side), and Federate (in the center), representing a HLA-interaction, HLA-object, and a HLA-federate respectively. The proxy elements are simply references to their respective target model

elements and are used in place of their targets to simplify the presentation of the model. The Federate element directly corresponds to any single instance of a simulation tool involved in the federation. As defined by the HLA standard [19], federates in a federation communicate among each other using HLA-interactions and HLA-objects, both of which are managed by the RTI. Interactions and objects, in an analogy with interprocess communications in operating systems, correspond to message passing and shared memory respectively.

In Fig. 2, the ParameterType attribute on the Parameter and Attribute elements defines the data type of that element (i.e. float, int, string). The Interaction and Attribute elements also support the HLA-defined attributes of Delivery and Order [19]. The primary attribute of a federate, as far as HLA-based synchronization is concerned, is its Lookahead, the interval into the future during which that federate guarantees that it will not send an interaction or update an object. The Federate as shown in Fig. 2 can correspond to various types such as Java, C++, Matlab/Simulink etc. The language elements StaticInteractionPublish, StaticInteractionSubscribe, StaticObjectPublish, and StaticObjectSubscribe, represent primitives necessary to model federates publishing and subscribing interactions, objects, and attributes [29]. A detailed description can be found in [29–31]. With these elements a designer is able to completely specify the integration model of the entire federation and its constituent simulation engines.

4.2. NCSWT model-based approach

The NCSWT MIL is an extension of base HLA meta-language. The major differences between NCSWT tool and the base HLA meta-language are the following:

1. We introduced a new federate to model the ns-2 simulator in the HLA framework and we extended the existing Matlab federate in the base HLA meta-language to specifically capture the design of a NCS.
2. We also introduced new interactions to specifically describe the exchange of information in NCS.
3. We developed two additional DSMLs that refine the NCS base model from NCSWT MIL by providing specific modeling concepts to model the dynamics of the control design and network design of the NCS.
4. We also provide model interpreters that automatically generate deployable system models for the control design and network design as well as glue code for run-time execution from the DSMLs.

Fig. 3 shows the model-based design architecture for NCSWT. The figure shows the design-time models used to define the NCS and its components using representative models in order to enable the simulation of a NCS. Fig. 3 shows the three DSMLs, NCSWT MIL, CDML and NDML utilized in our framework. The block "MATLAB control model" represents the Matlab/Simulink control models which are provided as parameters in the CDML. Likewise, the block "ns-2 Topology model"

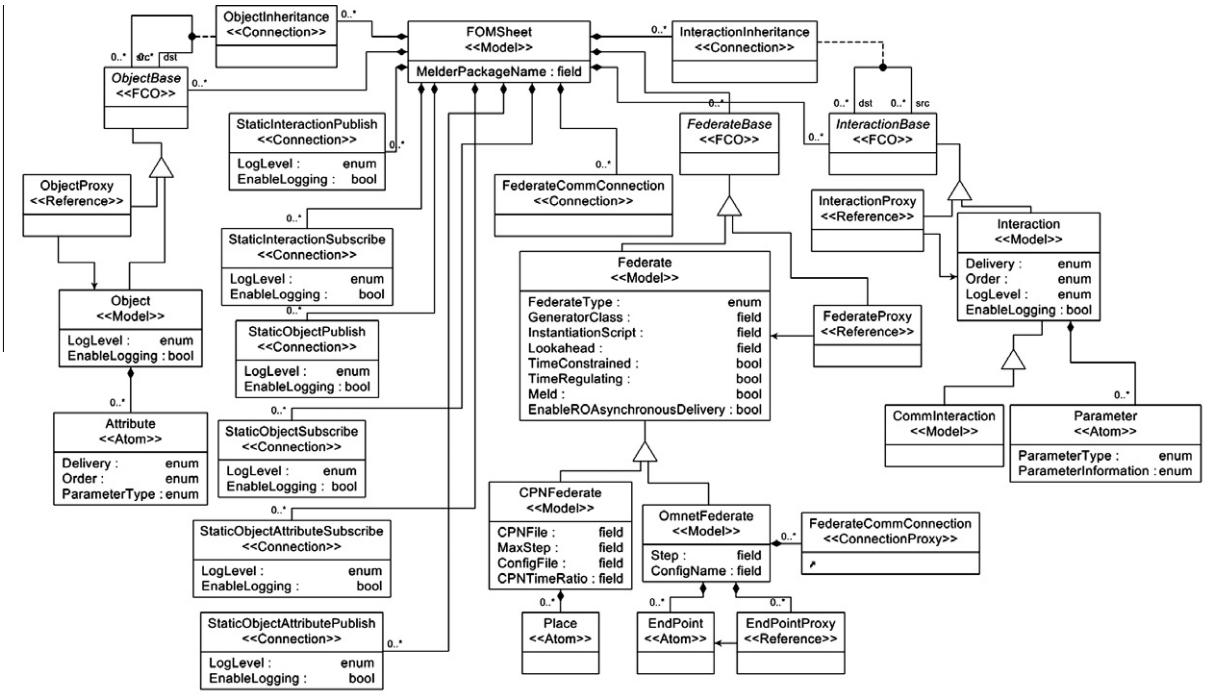


Fig. 2. Base HLA meta-language.

represents the network topology of the NCS provided as parameters in the NDML. From the three DSMLs, a set of reusable model interpreters are used to generate system models, configuration files, deployment models and interface scripts for the simulation of NCS. We provide a description of the DSMLs, and we use an example NCS to define a model for each DSML.

4.2.1. NCSWT model integration language

The NCSWT Model Integration Language (NCSWT MIL) provides the modeling primitives to specify the NCS in terms of elements in the HLA framework, such as federates representing the simulators, interactions representing the communication between the simulators, and parameters exchanged in the interactions, 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 defines the component of the NCS in terms of federates, interactions and the parameters. The executables for configuring the run-time environment for a specific NCS are generated from the base architecture model. The NCSWT MIL describes the tight coupling between the control and network design views of the NCS by defining how the two design views interact. This tight coupling ensures the consistency of the data semantics among the design views. Fig. 4 shows the NCSWT MIL meta-model.

1. *Federates:* The federates shown in the NCSWT MIL meta-model represent the elements for describing the simulators used in the NCS simulation based on the HLA framework. The NCSWT MIL models two main types of federates: the ND federate and the CD federate.
   (a) *ND Federate:* This federate models the software component for interfacing the network layer simulator, ns-2, with the RTI.
   (b) *CD Federate:* This federate models the software component for interfacing the control layer simulator, Matlab/Simulink, with the RTI. The CD Federate extends the previously-existing Matlab federate in the base HLA Meta-language [30] with additional attributes. The additional attributes AppID, AssociatedNodeId and AssociatedSystem are used in the definition of parameters in the control and network layers of the NCS.
2. *Interactions:* In the HLA framework, the information exchanged between federates are defined using interactions. In the NCSWT MIL, we introduce three main types of interactions in order to capture the main type of information exchange that can exist in a NCS. The three types of interactions that can be modeled in NCSWT MIL are Network interaction, Crosslayer interaction, and Control Design interaction.
   (a) *Network Interaction:* This interaction type enables the modeling of packets or information exchanged between control design components over the communication network. This includes, for example, the exchange of information between a plant and controller through the communication network such as sending plant outputs to the controller and sending control signals to the plant. In the NCSWT MIL, this interaction type always occurs as a pair with one

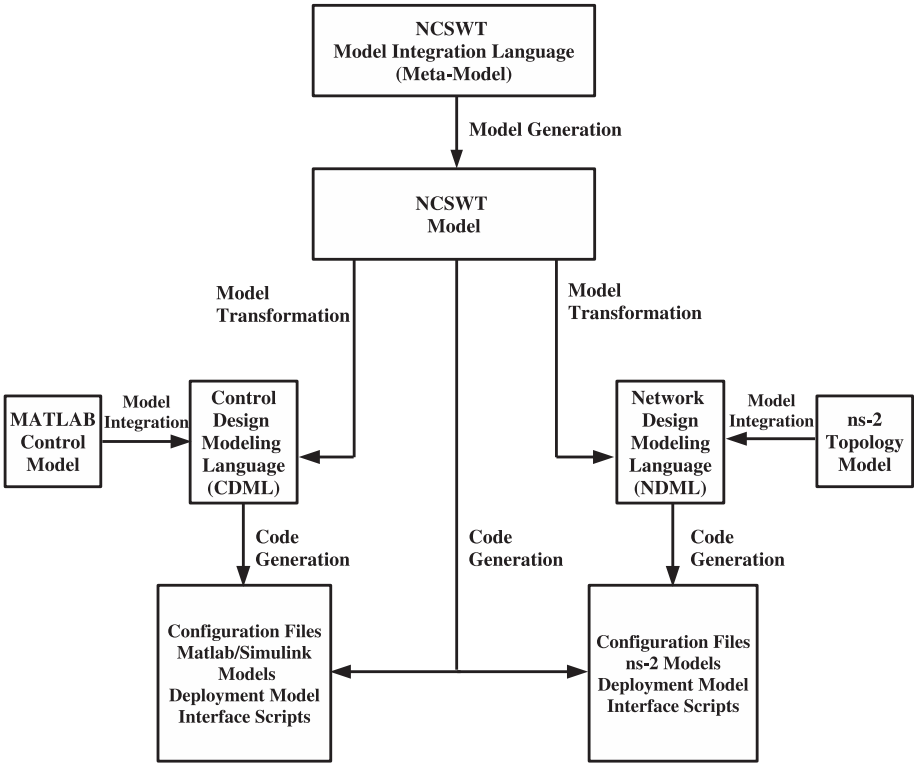


Fig. 3. Model-based design architecture for NCSWT.

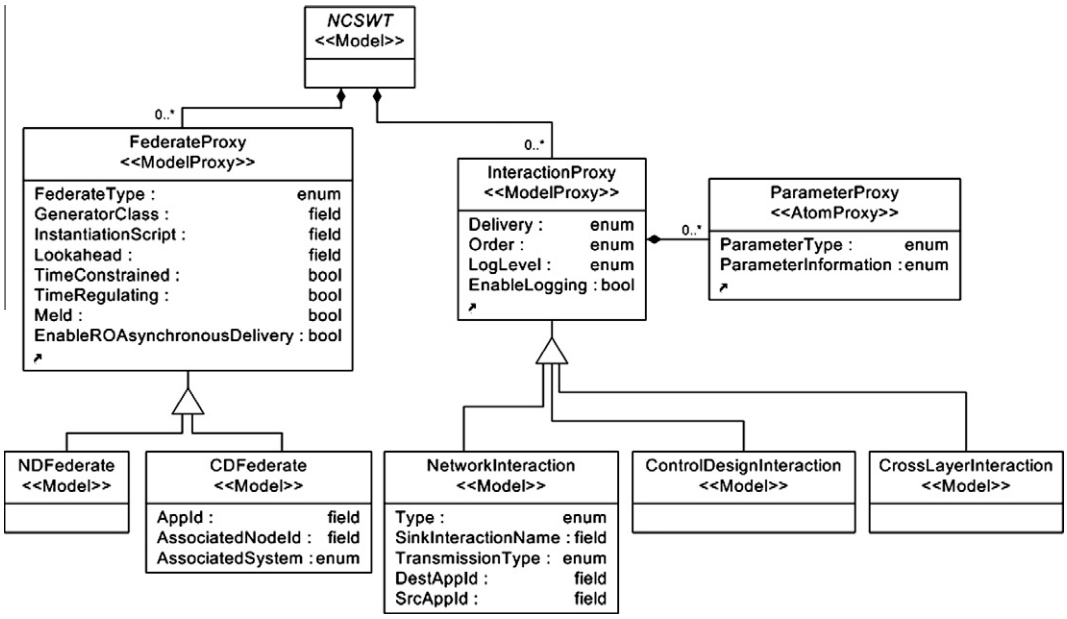


Fig. 4. NCSWT meta model.

interaction acting as the source and the other acting as the sink. The source interaction represents communication from a CD federate to the ND federate while the sink interaction represents the communication from the ND federate to another CD federate, which is the destination. This implies that a source network interaction must have a corresponding sink interaction and vice versa.

(b) *Crosslayer Interaction:* This interaction type allows for the modeling of information exchange between the network and application layers of a network protocol stack. It models the local communication between control design components and their respective local network interface. A typical example of crosslayer interaction is when a control design component queries its network interface for current network condition such as bit rate or loss rate etc. Such information can be used by the control design component to evaluate the quality of service of the network in order to implement a desirable control law or for other objectives.

(c) *Control Design Interaction:* This interaction type allows the modeling of the information exchanged between control design components that are transmitted or received by means other than the communication network. A typical example where this type of interaction is needed is modeling a radar sensor on a UAV for detecting the proximity of an obstacle or another UAV in its vicinity. This information is typically processed at the application (control design) level without being sent over the communication network.

Example. We introduce a NCS example to illustrate the NCSWT MIL. Fig. 5 shows a linear-time invariant continuous plant controlled by a proportional derivative (PD) digital controller over a 802.11b wireless network. The objective of the NCS is for the plant to track a reference velocity profile based on feedback information from the controller. The plant output is periodically sent to the digital controller while the control law is periodically sent to the plant from the digital controller. Fig. 6 shows a model of the NCSWT MIL for the example scenario. The Plant and the Controller are modeled as CD federates each representing an instance of the Matlab/Simulink simulation tool for each component. The communication network is modeled as a ND federate representing the ns-2 simulator. The blocks PlantIn, PlantOut, ContIn and ContOut are network interactions representing the information exchange between the plant and the digital controller over the network. In this example, the PlantOut, a source interaction, represents sensor information from the Plant, and ContIn is the sink interaction that eventually receives the sensor information after it goes through the network. Similary, the interaction ContOut is the source interaction representing the control signal from the Controller and PlantIn is the corresponding sink interaction that eventually receives the control signal after it goes through the network.

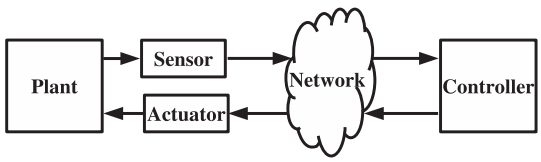


Fig. 5. Example networked control system.

4.2.2. Control design modeling language

The Control Design Modeling Language (CDML) defines the modeling elements for describing the dynamic behavior of the system components of the NCS. Fig. 7 shows the metamodel of the CDML, which defines control design concepts representing the control design view of a modeled NCS. The main components are the system type and the type of connection between system model components in the NCS. The system type models the dynamical behavior of the components of the NCS and can be modeled as one of three types.

1. *Plant:* This models the dynamics of the system to be controlled.
2. *Controller:* This models the control law or algorithm to modify the behavior of the plant to behave in a desired manner.
3. *Agent:* This essentially models a dynamic component which can neither be categorized specifically as a plant or a controller. For example, an UAV in a network of UAVs whose individual behavior is controlled by its neighbors.

A model created from the CDML is a refinement of the base architecture model of a NCS, created by the NCSWT MIL, with the details regarding the dynamics of the control system defined. In order to ensure consistency with the base architecture model defined in the NCSWT ML, a model transformation is used to transform a base architecture model directly to a base model in CDML. Then the control design concepts in CDML are used to define the dynamics of the components in the NCS.

It should be noted that CDML does not re-implement all of the Matlab/Simulink syntax and blocks, it provides an interface for generating and building specified control design models based on a user-defined library of Simulink blocks and inputs for a NCS. In CDML, a set of modeling primitives and attributes such as T_s (sampling time), ModelName and ModelLibraryName etc., can be used to specify the model and parameters that define the control system components. From the defined parameters in CDML, an integrated interpreter in CDML generates Matlab code which when executed builds the desired Simulink models for implementing the control system with integrated HLA-based interfaces for run-time simulation in the HLA framework.

Example. In order to illustrate the CDML, we use the NCS example defined in Fig. 5. Fig. 8 shows a model of the CDML describing the control design model of the example. Unlike the model, in Fig. 6, the Plant and Controller in this case represent the dynamics of the plant and the digital controller respectively. Using a set of modeling attributes defined in CDML, a user can specify the user-defined Matlab/Simulink model and parameters that define the dynamic behavior of a control system component.

4.2.3. Network design modeling language

The Network Design Modeling Language (NDML) 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. Fig. 9 shows the meta model of the network design modeling language. From Fig. 9, the main components are node, application, transport agent, link and the various connection types that exist between the components.

1. *Node:* The node component represents the host or a computing unit on which an particular application (control design component) is running.
2. *Application*: This models the actual application that runs at the top of the network stack. In our case this is an abstraction of the control design components.
3. *Transport Agent*: This models the protocols or agents for delivering messages from the application layer of the source to that of the destination such as UDP or TCP.
4. *Link*: This models the link layer of the network. It defines the path for delivering packets from a source node to a destination node.

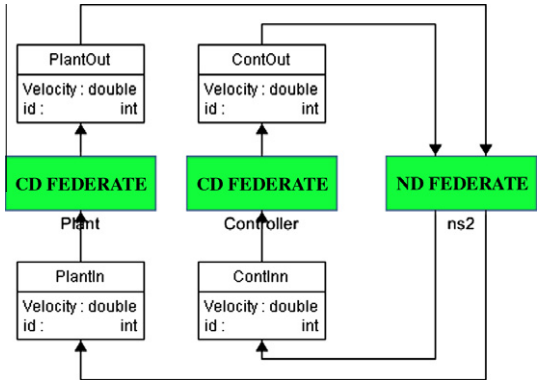


Fig. 6. NCSWT model.

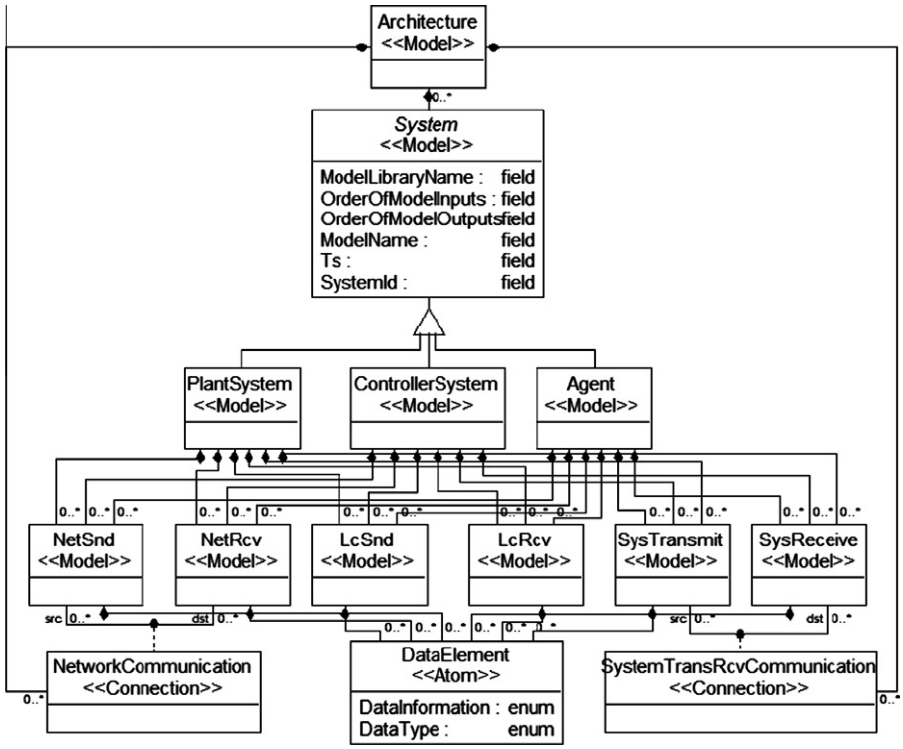


Fig. 7. Control design meta model.

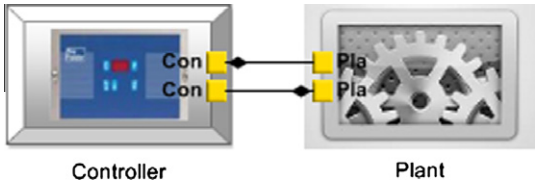


Fig. 8. Control design model.

It should be noted NDML does not re-implement the ns-2 syntax and simulation semantics. NDML essentially provides an interface language for defining the network interactions between the ns-2 simulation and the Matlab simulation. Examples of such interactions include the controller/plant deployment on the network node, and traffic types etc. The ns-2 syntax and semantics that are internal to networking simulation (e.g., packet scheduling, routing) will not be reflected in NDML. We rather rely on the TCL language of ns-2 for the network internal simulation configuration.

Similar to CDML, a model transformation is used to transform the base architecture model in NCSWT MIL directly to a base model in NDML in order to preserve the consistency of the models in the different design views. Then the network primitives defined in NDML 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 using an integrated code generator in NDML based on the defined model parameters for deployment in ns-2 for the execution of the NCS simulation.

Example. In order to illustrate the NDML, we use the NCS example defined in Fig. 5. Fig. 10 shows a model of the NDML describing the network design model of the example. This model specifies the transport agents, the network topology and other network properties required for the exchange of information between the control design components over a network. Unlike the model shown in Figs. 6 and 8, the Plant and Controller in this case represent the network applications. The TPAgent1 and TPAgent2 represent the transport agent models attached to the network hosts, Node1 and Node2 respectively. Using a set of modeling primitives and attributes in NDML, a user can specify the loss model of network links to simulate the desired network dynamics.

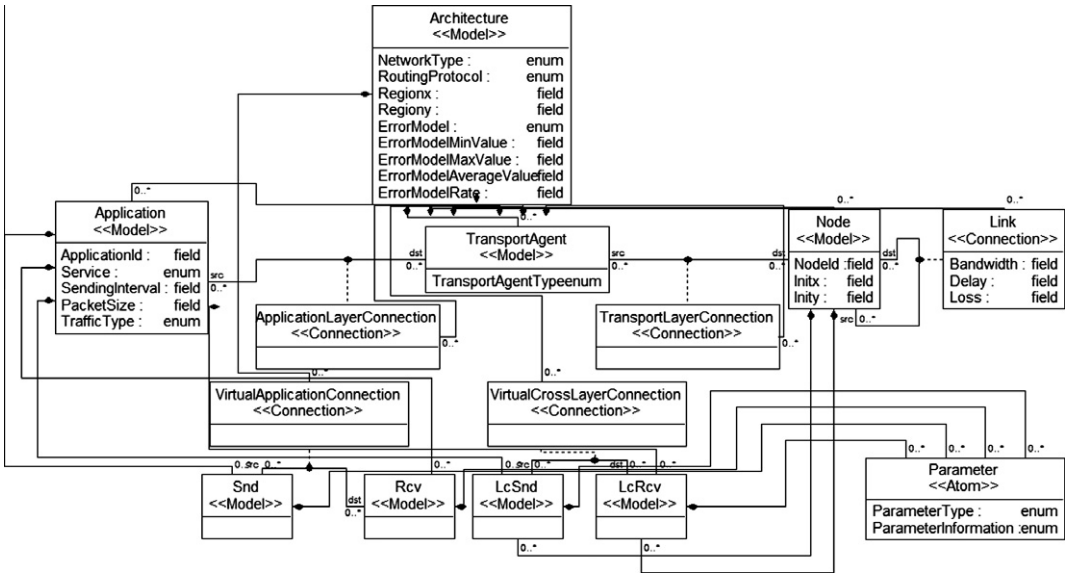


Fig. 9. Network design meta model.

4.3. Design flow for the NCSWT model-based approach

Fig. 11 shows the design flow for the NCSWT model-based approach. The number depicted on the bottom right of each of the blocks represents the design stage of the model-based approach.

- *Step 1:* In this step, a user defines the NCS to be simulated in the NCSWT MIL using HLA concepts that capture the control components and network components of the NCS as well as the information flow between the components. Using a set of integrated interpreters, two model transformations are executed on the NCSWT MIL to generate a CDML model and a NDML model for the definition of the control design and network design components. Additionally, configuration scripts and glue code required for the run-time simulation of the NCS are also generated from the NCSWT MIL model.
- *Step 2:* In this step, a user designs the control components of the NCS in Matlab/Simulink. Each individual component of the NCS that is to be simulated separately is stored in a user defined library. The reference name of the component model and the library name in which it is stored are used in the CDML. If the control component is a standard Simulink block that can be located in the Simulink library, the user determines the reference name of the block in the library as this is used in the CDML in order to build the NCS model.
- *Step 3:* Depending on the complexity of the network, a user can specify a network topology of the NCS in ns2 using a third-party tool such as GT-ITM for realistic Internet topology generation [32]. The reference name of this topology can be provided as an input to the NDML. Alternatively, a medium sized network topology can be modeled directly in NDML.

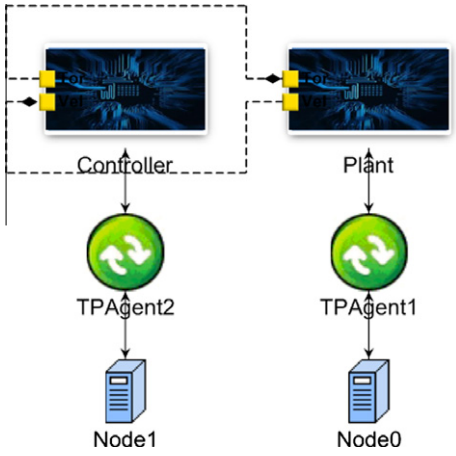


Fig. 10. Network design model.

- *Step 4:* This step involves the modeling of control components using CDML. In CDML, starting from the transformed model from the NCSWT MIL, each of the control design components are captured through the use of attributes defined in the CDML. These attributes specify the user library containing the control component Simulink models as well as the reference block names for the components created in Step 2. An integrated interpreter traverses the CDML instance model and generates Matlab code which when executed generates Matlab/Simulink models for the control design components integrated with the HLA-based interfaces required for the NCS simulation.
- *Step 5:* In the NDML, starting from the transformed model from the NCSWT MIL, a user specifies the network components such as the transport agent, node and application agents for the NCS. These components together with their parameterized attributes capture the underlying network communication features for the NCS. A set of integrated interpreters traverses the model and generates tcl for the NCS model. Also, the HLA-interface scripts required for the integration of ns2 for the run-time simulation is also generated file.
- *Step 6:* The final step involves the deployment of the generated models, glue code and configuration files from the subsequent steps in the run-time environment to facilitate the actual simulation of the NCS using our framework.

5. NCSWT run-time components

The run-time components of the NCSWT are shown in Fig. 12. These components represent the main software components and interfaces for the realization of a NCS simulation using the HLA framework. These components include the simulators (Matlab/Simulink and ns-2), 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.

5.1. Run-Time Infrastructure (RTI)

The RTI, an implementation of the HLA standard, manages the communication between different federates. Using interactions, federates communicate between each other through the RTI [33]. 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 [33].

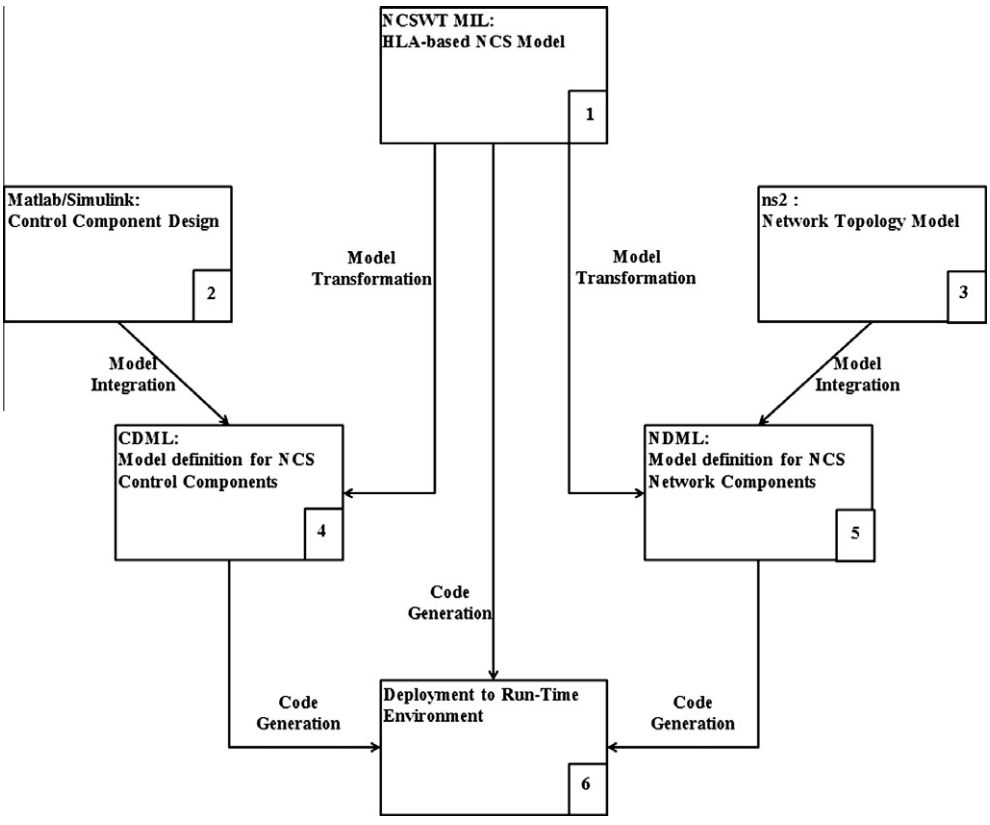


Fig. 11. Model-based design flow for NCSWT.

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 ND federate is a software component that interfaces the ns-2 simulator with the RTI. The RTI provides a set of programmable application interfaces to instantiate a federate and for the federate to communicate with the RTI.

We briefly describe the NCSWT run-time services provided by the RTI.

5.1.1. Time management

In an 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 logical time. The RTI ensures the accurate progression of time through the use two main basic operations, the time advance request (TAR) and the time advance grant (TAG). In order for a federate to progress its logical clock during a simulation run, it must send a TAR to the RTI which then determines based on the current clocks of all the other federates in the federation whether to send a TAG for the federate to proceed. The RTI chooses the smallest of all the federates' TAR times, if a federate TAR time is larger than the granted TAG, it will block and it will only proceed its simulation when its TAG is successful [28].

5.1.2. Data communication and coordination

The RTI uses a publish-and-subscribe mechanism for passing messages through the federation in order to ensure the accurate data communication and coordination between the federates [28]. The type of messages exchanged between the federates are determined by the interactions and the interaction parameters defined in the NCSWT MIL. In the publish-and-subscribe mechanism, the sender of the interaction, known as the publisher, publishes the interactions to the RTI making it available to any receiver, known as the subscriber, registered to receive those kind of interactions. The RTI assures timely delivery of the interactions to all subscribing federates.

In addition to these services, in our framework the RTI provides additional services for monitoring the progression of a simulation during run-time.

5.2. Federates

In order to participate in a federation, the ND and CD federates utilize the services provided by the RTI. We briefly discuss how each of the federates utilize these services.

5.2.1. ND federate

In order to participate in a federation, a ND federate, which interfaces the communication network simulated in ns-2 to RTI, leverages a set of generic classes defined by completely reusable C++ code. The reusable C++ code provides all of the fundamental RTI integration requirements such as converting between ns-2 defined types and RTI types, encapsulating and interfacing with the RTI for initializing the federate, synchronizing the simulator's clock and managing the publish-and-subscribe relationship with other federates.

The design-time models facilitate the integration of the ND federate in an HLA based federation. In our framework, a ND federate can be defined in the NCSWT MIL model, along with all its associated interaction types and parameters. This definition directly integrates the network design components simulated in ns-2 with an HLA-federation. From the NCSWT MIL, a GME interpreter is used to generated C++ files used during the run-time simulation of the NCS. The generated files leverage the reusable generic C++ classes together with the integrated interactions and parameters specification for the ND federate defined in the NCSWT MIL in order to participate in a federation.

Fig. 13 shows a time synchronization example with the ND federate. Because the ns-2 simulator uses a discrete event model of computation, time synchronization can be incorporated along with event scheduling. In particular, the C++ source code for the ns-2 simulator scheduler is modified to implement the time synchronization mechanism defined by the RTI. The scheduler blocks the scheduling of new events until it receives an adequate time advance grant (TAG) from the RTI. The ns-2 scheduler submits a time advance request (TAR) once it is ready to execute an event scheduled for at a time later than the latest TAG. An initialization TAR is passed from ns-2 to the RTI, and once a TAG is received, the ns-2 scheduler can unblock

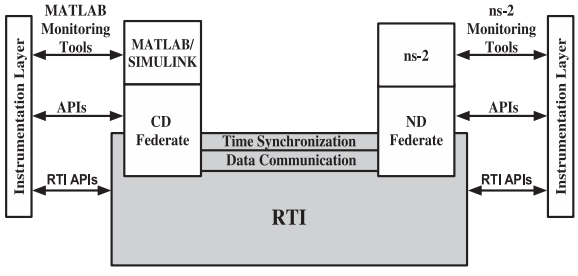


Fig. 12. Run-time components.

and run until it reaches a synchronization event (an operation requiring synchronization with the RTI). At that point, a TAR is submitted to the RTI and ns-2 blocks. It is possible that the newest TAG will be less than the most recent TAR (i.e. $t_2 < t_1$), so ns-2 only executes events scheduled for times less than or equal to the latest TAG. Since our modification is made to the base class scheduler, our approach naturally supports the different types of Calendar Schedulers that are available in ns2.

For the ND federate's data communication, the communication network simulated in ns-2 uses a set of function calls defining the interactions associated with the federate, together with the reusable generic C++ classes to send and receive interactions using the publish and subscribe mechanism provided by the RTI. The function calls are generated from a set of GME interpreters integrated in the NCSWT MIL and NDML. The interactions, which are modeled in the NCSWT MIL, represent the data exchanged between the ns-2 simulator and Matlab/Simulink.

5.2.2. CD federate

In order to participate in a federation, a CD federate, which interfaces the control design components simulated in Matlab/Simulink to RTI, leverages a set of generic classes defined by completely reusable Java code. The reusable code provides all of the fundamental RTI integration requirements such as converting between Matlab/Simulink types and RTI types, encapsulating and interfacing with the RTI for initializing the federate, synchronizing the simulator's clock and managing the publish-and-subscribe relationship with other federates [29].

In our framework, a CD federate can be defined in the NCSWT MIL model, along with all its associated interaction types and parameters. This definition directly integrates the control design component simulated in Matlab/Simulink with a HLA-federation. From the NCSWT MIL, a GME interpreter is used to generated Java and Matlab/Simulink files used during the runtime simulation of the NCS. The generated files leverage the reusable generic classes [29], together with the integrated interactions and parameters specification for the federate defined in the NCSWT MIL, in order to participate in a federation.

The time synchronization mechanism for the CD federate is simulated in Matlab/Simulink using a Simulink S-function block integrated in Simulink model of each control design component. Each control design component, using function calls in the S-function blocks, implements the time synchronization mechanism provided by the RTI [29]. These function calls implement the TARs/TAGs mechanism for each of the Matlab/Simulink component. Using these function calls, the component can request the advancement of the simulator's logical clock and Using these function calls, the component can request the advancement of the simulator's logical clock and the simulator's execution is allowed to proceed only when the RTI grants the advancement of the simulator's logical clock.

For a CD federate, each corresponding control design component also uses the Simulink S-function block to either publish or subscribe interactions in order to send or receive data. The control design component, using function calls in the S-function blocks, implements the sending or receiving of interactions from the RTI.

The CDML modeling language, automatically generates the Simulink model for the corresponding control design component with the integrated blocks containing the S-function calls for implementing the data communication and time synchronization mechanisms.

6. Implementation overview

The NCSWT tool is intended to provide users with a flexible, extensible and convenient tool for simulating NCS. The simulation of a NCS using the NCSWT tool requires two major steps. The first step involves the modeling of the NCS and the generation of all the necessary models, configuration scripts 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 4. 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. Fig. 14 shows pictorial representation of NCSWT Implementation for the described simulation steps.

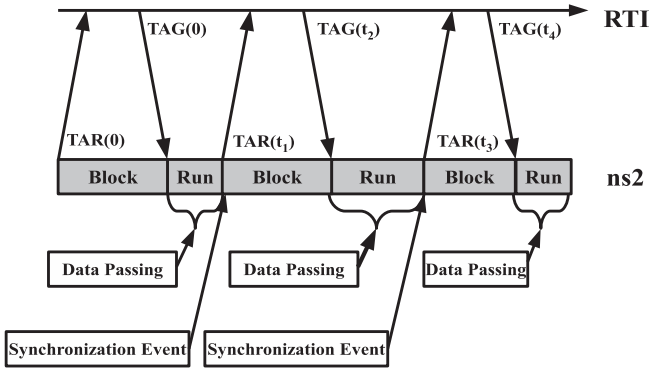


Fig. 13. Time synchronization between the NS-2 federate and the RTI.

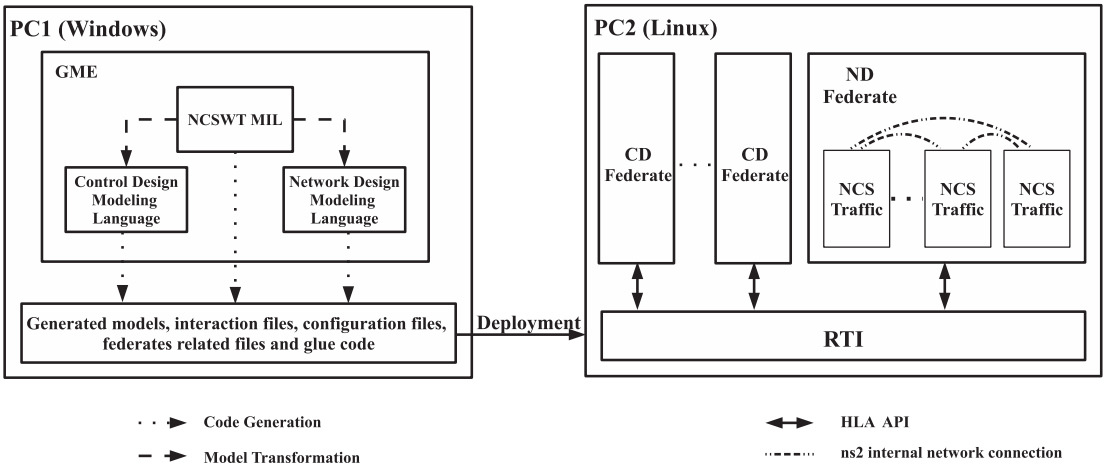
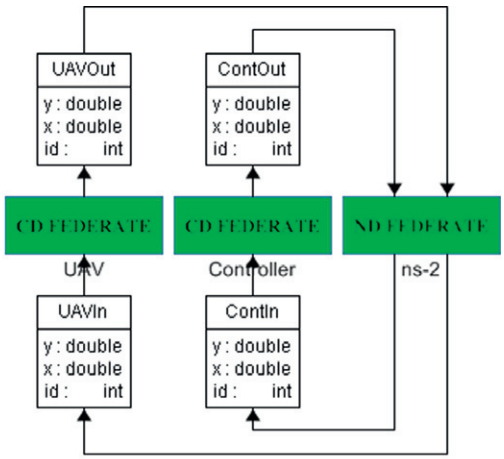
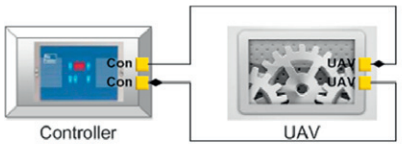


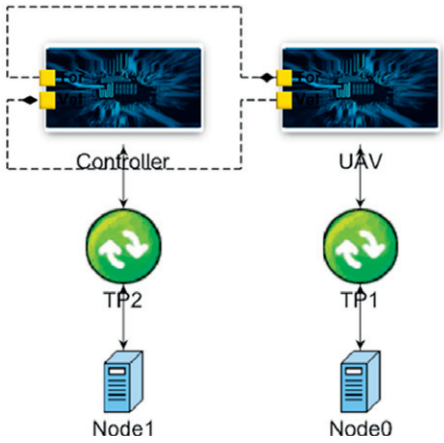
Fig. 14. NCSWT implementation overview.



(a) NCSWT MIL Model



(b) CDML Model



(c) NDML Model

Fig. 15. Design-time models for networked aerial vehicle.

7. Case studies

We present two case studies to demonstrate our tool as well as show how it can be used to evaluate the impact of network effects such as time-varying delays and packet losses on the overall performance of a NCS.

7.1. Networked unmanned aerial vehicle

This NCS is comprised of an unmanned aerial vehicle (UAV) [34] controlled by a proportional derivative (PD) digital controller over a 802.11b wireless network. The networked digital controller, a proportional-derivative (PD) controller, is designed to enable the UAV to track a desired reference position trajectory. The main objective of this case study is to demonstrate the use of the NCSWT tool to simulate the impact of various network effects on the NCS. The UAV and the digital controller are modeled in Matlab/Simulink while the wireless network is modeled in ns-2. We consider three main scenarios: (1)Nominal case, (2) Scenario with a lossy network, and (3) Scenario where multi-hop relay is employed for packet delivery. Fig. 15, shows the design-time models for the Networked Aerial Vehicle.

7.1.1. Nominal case

In this experiment, we simulate the NCS composed of the UAV and a digital controller communicating through a single hop wireless network. The UAV and the networked controller are located within the transmission range of each other and there are no additional network effects such as packet losses and delays. The sampling period of the networked controller is 0.1 s. Fig. 16a shows a plot of the UAV x and y positions as well as the reference position trajectory. The plant tracks the reference trajectory so closely that the difference is imperceptible in the figure. Fig. 16b shows the end-to-end delay plot for the Nominal case with the sampling period denoted by the horizontal line at 0.1 s. It can be seen that the delay is much less than the sampling period, so it is to be expected that the plant is able to follow the reference signal closely.

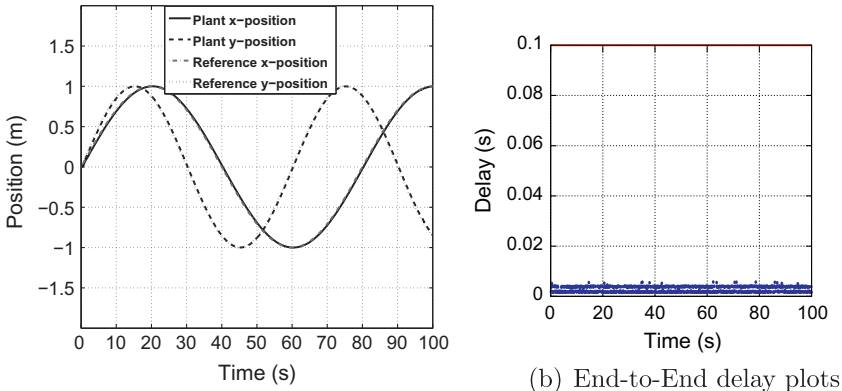


Fig. 16. Plots for the nominal case of the networked UAV.

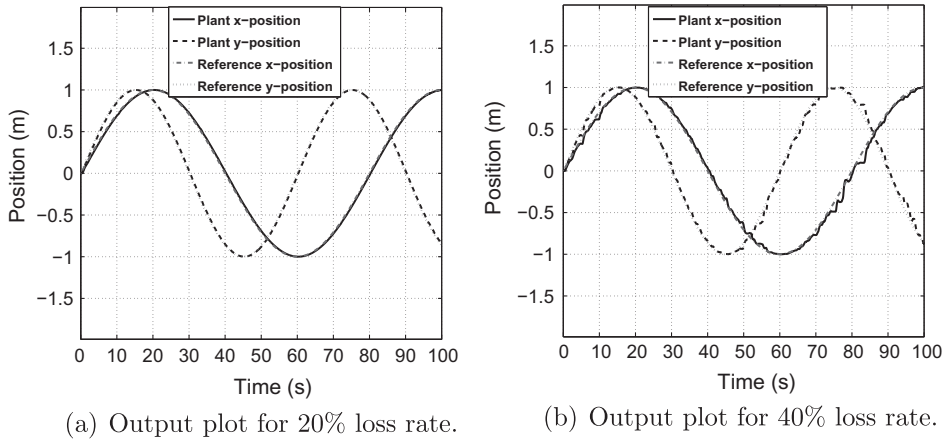
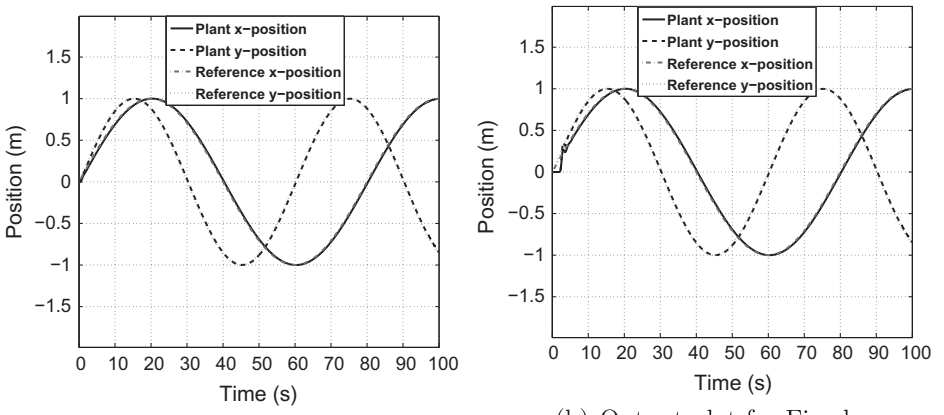


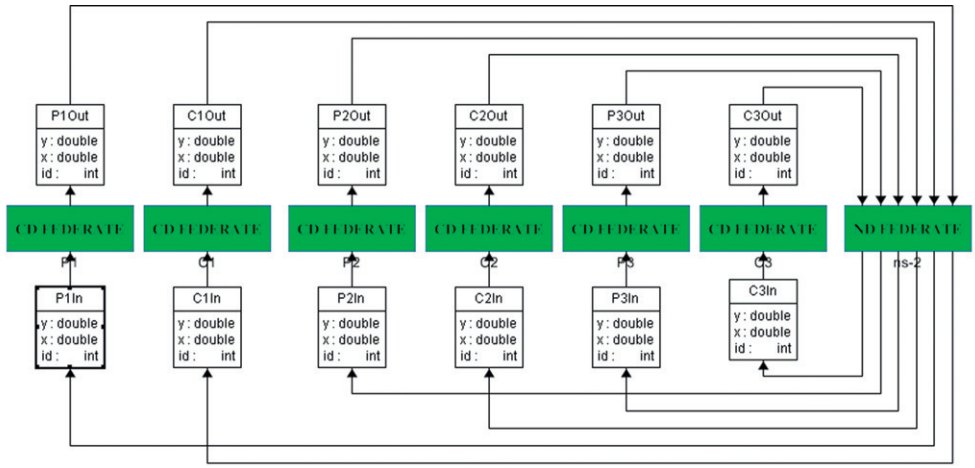
Fig. 17. Plots of UAV trajectory for packet loss rates.



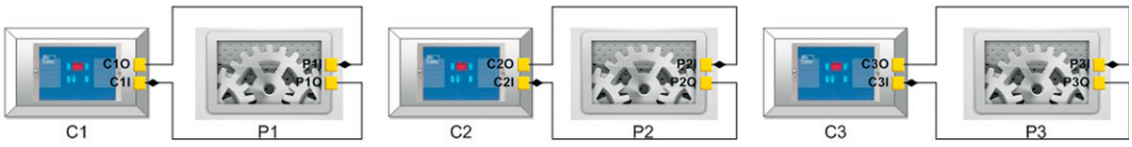
(a) Output plot for Three hops.

(b) Output plot for Five hops.

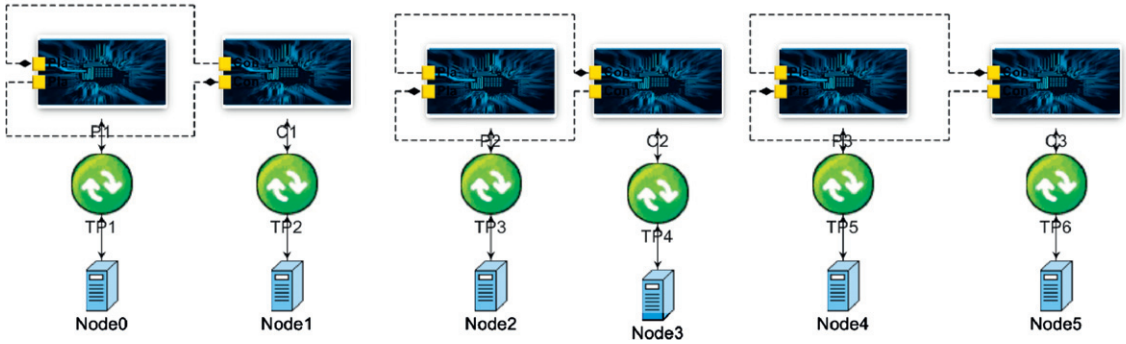
Fig. 18. UAV trajectory for multi-hop communication.



(a) NCSWT MIL Model



(b) CDML Model



(c) NDML Model

Fig. 19. Design-time models for INCS.

7.1.2. Scenario with a lossy network

Wireless network communication can be unreliable, so in this experiment we demonstrate the impact of packet losses on the performance of the NCS when packets are lost in the communication channel of the network. We simulate a lossy communication channel based on a uniform probability distribution [35]. The results for the loss rates of 20% and 40% are presented in Fig. 17. As the loss rate increases, the UAV trajectory strays further from the reference trajectories as can observed in Fig. 17 which would be expected.

7.1.3. Scenario where multi-hop relay is employed for packet delivery

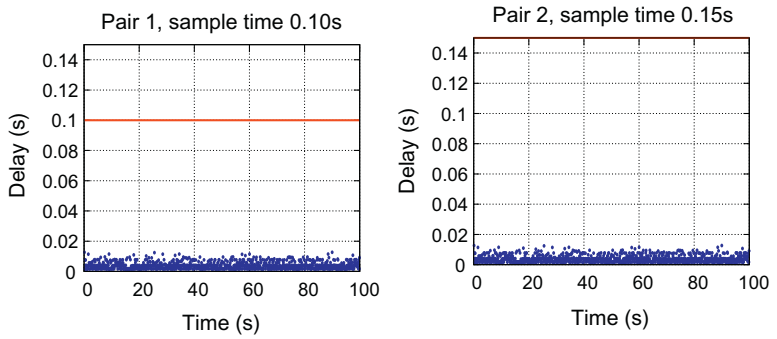
The direct connection of two nodes in wireless networks requires the two nodes to be within the transmission range of each other however, this may not always be possible. In order to enable the communication of two nodes that are outside of each other's transmission range, intermediate nodes can act as relays to route the packets to their final destination nodes. Such network architecture are called wireless multi-hop networks. We use a chain topology with a static routing protocol to test the multi-hop scenario for the Networked UAV NCS. In this topology, nodes are formed in a chain structure with a fixed distance of 200 m between neighboring nodes. The plant node and controller node are located at the edges of the network such that they will need to transmit information through intermediate nodes.

Fig. 18, shows the simulation results for three and five chain networks. From Fig. 18, it can be seen (especially at the beginning of the trajectory plots) that as the number of hops increase, the UAV trajectory deviates further away from the reference trajectory due to the increased delay introduced by the number of hops between the plant and controller nodes.

7.2. Industrial Networked Control System (INCS)

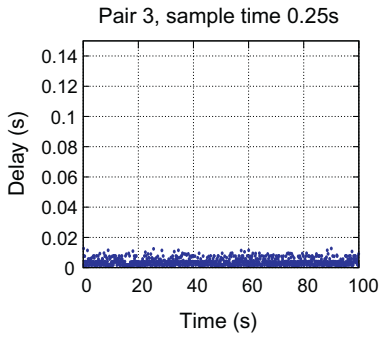
In this case study we demonstrate the ability of the NCSWT to handle asynchronous sampling times in large systems. Systems with asynchronous sampling rates can cause resource contention problems for the communication channel, so it is very important to accurately model the behavior. Additionally, since NCS can potentially be large in size it is also very important that the tool is able to handle NCS of varying sizes. By using the HLA communication standard, the NCSWT simulation tool appropriately handles the simulation of large NCS with various sampling rates as described in Section 5.

This case study is a typical industrial networked control system. It involves the simulation of three networked control systems working concurrently over the same wireless network. The networked control systems are denoted NCS1, NCS2 and NCS3, and they execute with the sampling times 0.1 s, 0.15 s and 0.25 s respectively. Each of the networked control sys-



(a) Delay plot for P1-C1 pair

(b) Delay plot for P2-C2 pair



(c) Delay plot for P3-C3 pair

Fig. 20. End-to-end delay plot for the nominal case scenario of INCS.

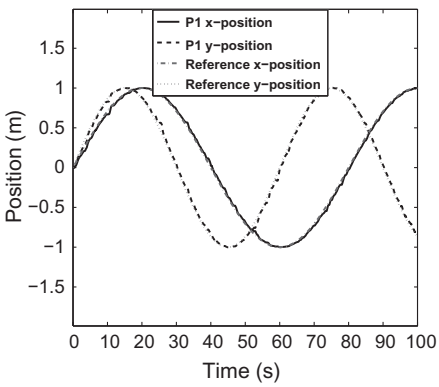
tems is composed of a plant system and a proportional derivative (PD) controller that controls the corresponding plant to behave in the desired manner. NCS1 is composed of the plant system P1 and controller C1; NCS2 is composed of the plant system P2 and controller C2 and NCS3 is composed of the plant system P3 and controller C3. The model for each of the plant systems, P1, P2 and P3, is an industrial robotic arm. The model for each of the controllers C1, C2 and C3 is identical except for the different sampling times. We consider two scenarios: (1) Nominal case, and (2) Scenario with network effects. Fig. 19, shows the design-time models for the INCS.

7.2.1. Nominal case

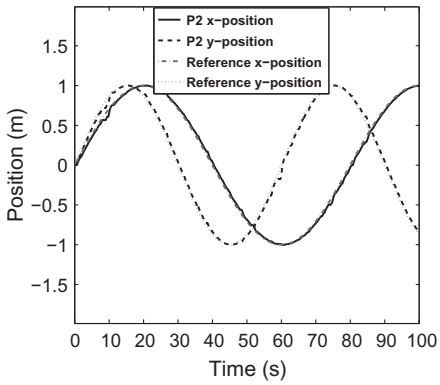
In this experiment, the three NCS are operating in a single-hop network without any additional network effects such as packet losses and delays. All six nodes use the same network channel. The reference trajectories for this experiment are essentially identical to the reference trajectory in the single UAV networked system in Section 7.1. Fig. 20 shows the end-to-end delay plots. The horizontal red line in each plot indicates the sampling time for each plant-controller pair. It can be seen that the delays for each of the NCS are less than their corresponding sampling period.

7.2.2. Scenario with network effects

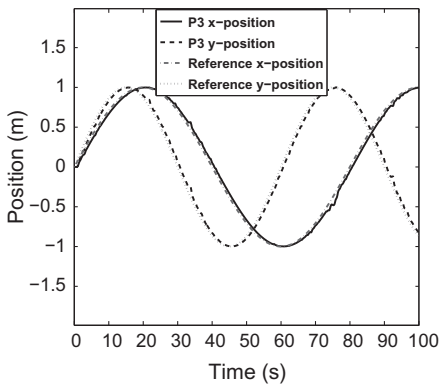
We simulate background traffic and packet loss in the wireless network to evaluate their impact of network effects on the overall performance of the INCS. Two nodes were introduced in the network to simulate background traffic while a uniform probability loss model [35] was used to simulate packet loss in the communication channel. Fig. 21 shows a plot of the outputs of the plants showing the effect of the background traffic in addition to a 30% loss rate. It can be seen that the simulated network effects affect the outputs of the plants as depicted by degraded performance in tracking the reference trajectories. Fig. 22 presents the network delay for packets transmitted between the plant and the controller for each of the three NCS. In the delay plots in Fig. 22, the horizontal red line in each plot indicates the sampling time for each plant-controller pair. The figure shows similar delay for most of packets in the system, and this is reasonable since all the six nodes share the same network channel and therefore contend for network resources.



(a) Output plot for Plant P1.

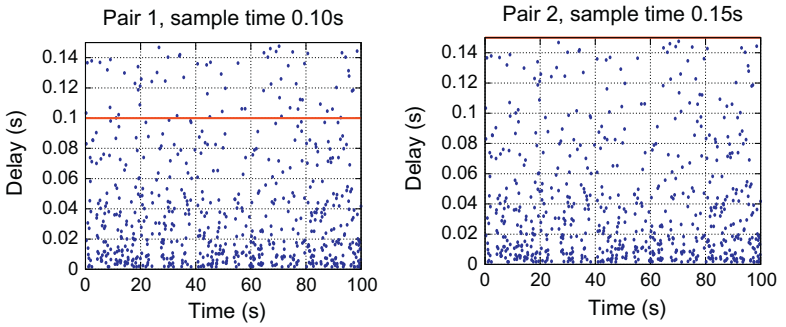


(b) Output plot for Plant P2.



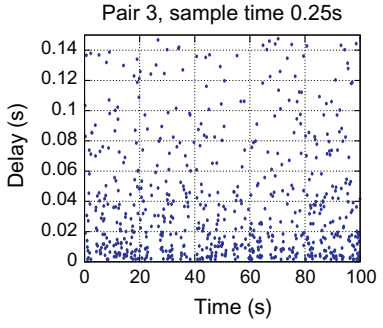
(c) Output plot for Plant P3.

Fig. 21. Output plots for the addition of background traffic and 30% packet loss in INCS.



(a) Delay plot for P1-C1 pair

(b) Delay plot for P2-C2 pair



(c) Delay plot for P3-C3 pair

Fig. 22. End-to-end delay for the addition of background traffic and 30% packet loss in INCS.

8. Evaluation

In order to build the NCSWT tool, the software packages shown in Table 2 are needed. Matlab/Simulink and ns-2 are used for the simulation of the control system and communication network 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. Universal Data Model (UDM) is utilized in the model transformations from the NCSWT MIL to CDML and NDML. 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. The software packages ns-2, Portico 1.0.2, GME, UDM and Eclipse are freely available, while Matlab/Simulink and Microsoft Visual Studio are proprietary tools.

Table 2
Required software packages.

1. Matlab/Simulink, www.mathworks.com
2. ns-2, http://isi.edu/nsnam/ns
3. Portico 1.0.2, www.porticoproject.org
4. Generic Modeling Environment (GME), www.isis.vanderbilt.edu/Projects/gme
5. Universal Data Model (UDM), http://www.isis.vanderbilt.edu/tools/UDM
6. Microsoft Visual Studio 2008 or later, www.microsoft.com/visualstudio
7. Eclipse, www.eclipse.org

Table 3
Generated code for networked UAV case study.

Files	Size
1. Matlab models	100 KB
2. Matlab glue code	132 KB
3. ns-2 model and topology scripts	20 KB
4. ns-2 glue code	160 KB
5. Federation startup script	4 KB

Table 4
Time efficiency for networked UAV case study.

Scenarios	Actual duration (in min)
Nominal	5.5
Packet losses	
20%	8.4
30%	11.4
40%	13.5
Multi-hop network	
3 hops	17.4
4 hops	22.2
5 hops	26.2

Table 5
Time efficiency for MANCS case study.

Scenarios	Actual duration (in min)
Nominal	10.0
Background traffic and 30% packet loss	
3 hops	17.4
4 hops	22.2
5 hops	26.2

NCSWT allows a user to rapidly reconfigure their experimental setup for the simulation of a NCS. For example, in order to evaluate the impact of network effects on a NCS, the configurations in the NDML is modified to the desired network configuration setup and then updated code is generated from the network model without modifying the control models. Similarly, if the dynamics of the control system is changed, the models in the CDML are modified to reflect the changes while maintaining the same network configurations, and then updated control models are generated.

If the information passed between the control system and the network is changed, the NCSWT MIL model of the NCS needs to be modified to reflect the change. For example, when an additional parameter or new information type is introduced in the NCS, a new interaction type can be added to the existing NCS model defined in the NCSWT MIL and the new parameter can be added to it. If the new parameter needs to be added to an already existing information exchange type, the parameter can be added to the existing interaction reflecting the information type that is exchanged between the federates. In a different case, if a new control component is introduced to the NCS, a new CD federate needs to be added to NCSWT MIL to reflect the introduced control design component and the required interactions are defined to reflect the communication of the new federate with the other federates. After the modifications to NCSWT MIL model of the NCS, update code and models are generated to reflect the changes to the model. In the case we need to modify the communication exchange between control components by adding new interactions or introduce additional components by adding federates, we use NCSWT MIL to add the necessary components and essentially repeat the NCS design process as described in Section 4.

We demonstrate the design-time efficiency for the Network UAV example. Recall, this NCS involves the digital control of an unmanned aerial vehicle (UAV), representing the plant, over the 802.11b wireless network to track a desired trajectory. For the design-time efficiency, we consider the amount of code that is automatically generated for simulating the NCS. Table 3 provides a summary of the size of code and models that are automatically generated from the design-time models. For the run-time efficiency, we consider the actual time it takes to simulate the NCS. Table 4 shows the time durations for simulating the NCS in various multi-hop network topologies and in the presence of network uncertainties such as packet loss and time varying delays. The time durations shown in Table 4 are the actual times required to run 100 seconds of logical simulation time.

We also provide the run-time efficiency of the multi-agent NCS (MANCS) example. Similar to the networked UAV case, the time durations shown in Table 5 are the actual times required to run 100 s of logical simulation time.

9. Conclusion and discussion

The design and analysis of NCS is a critical task due to the complex interactions and uncertainties introduced by network effects. Simulation is a powerful technique in evaluating various NCS models and the impact of network effects on the overall system performance. In this paper, we present an integrated modeling and simulation tool, NCSWT, for NCS. We describe 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. We demonstrate the capabilities of the tool using case studies and also provided an evaluation of tool.

The current version of the NCSWT tool addresses only time-driven NCS systems. The support for the simulation of event-triggered NCS such as in [36] is an interesting subject of future extension. In addition to the provision of TAR which enables the simulation of time-driven NCs, the HLA also offers another time synchronization mechanism called Next Event Request (NER) [19]. This service enables the ability to synchronize a federate based on the occurrence of events rather than a pre-specified time intervals as in the case of TAR. This mechanism can be used for event-triggered control applications which involve the triggering of control computation based on the occurrence of an event such as the plant state exceeding a certain threshold.

Our use of HLA-based framework provides great opportunities in regards to extending the framework to other simulators. The approach is modular and extensible. Based on previous works in [29–31] as well as the NCSWT, which involve the integration of various simulators using HLA, the inclusion of other simulators such as ns3, opnet, and Modelica can follow a similar approach. The main challenges involve identifying the critical sections in the simulators to integrate the time synchronization mechanism and data distribution services offered by the HLA standard.

Acknowledgements

We would like to thank Himanshu Neema and Harmon Nine for their assistance in setting up the run-time environment. This work is supported in part by the U.S. Army Research Office (ARO W911NF-10-1-0005), the National Science Foundation (CNS-1035655, CCF-0820088) and Lockheed Martin. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government.

References

[1] P. Antsaklis, J. Baillieul, Special issue on technology of networked control systems, Proceedings of the IEEE 95 (1) (2007) 5–8.
[2] J. Baillieul, P. Antsaklis, Control and communication challenges in networked control systems, Proceedings of the IEEE 95 (1) (2007) 9–28.
[3] R. Gupta, M.-Y. Chow, Networked control system: overview and research trends, IEEE Transactions on Industrial Electronics 57 (7) (2010) 2527–2535.
[4] Y. Tipsuwan, M.-Y. Chow, Control methodologies in networked control systems, Control Engineering Practice 11 (10) (2003) 1099–1111.
[5] L. Schenato, B. Sinopoli, M. Franceschetti, K. Poolla, S. Sastry, Foundations of control and estimation over lossy networks, Proceedings of the IEEE 95 (1) (2007) 163–187.
[6] J.P. Hespanha, P. Naghshtabrizi, Y. Xu, A survey of recent results in networked control systems, Proceedings of the IEEE 95 (1) (2007) 138–162.
[7] M. Pajic, S. Sundaram, G. Pappas, R. Mangharam, The wireless control network: a new approach for control over networks, IEEE Transactions on Automatic Control 56 (10) (2011) 2305–2318.
[8] H. Gao, T. Chen, T. Chai, Passivity and passification of networked control systems, SIAM Journal of Control and Optimization 46 (4) (2008) 1299–1322.
[9] H. Li, Z. Sun, M.-Y. Chow, F. Sun, Gain-scheduling-based state feedback integral control for networked control systems, IEEE Transactions on Industrial Electronics 58 (6) (2011) 2465–2472.
[10] A. Onat, T. Naskali, E. Parlakay, O. Mutluer, Control over imperfect networks: model-based predictive networked control systems, IEEE Transactions on Industrial Electronics 58 (3) (2011) 905–913.
[11] Y.-B. Zhao, G.-P. Liu, D. Rees, Packet-based deadband control for internet-based networked control systems, IEEE Transactions on Control Systems Technology 18 (5) (2010) 1057–1067.
[12] M. Cloosterman, L. Hetel, N. van de Wouw, W. Heemels, J. Daafouz, H. Nijmeijer, Controller synthesis for networked control systems, Automatica 46 (10) (2010) 1584–1594.
[13] Y.-B. Zhao, G.-P. Liu, D. Rees, Design of a packet-based control framework for networked control systems, IEEE Transactions on Control Systems Technology 17 (4) (2009) 859–865.
[14] Matlab, The Language of Technical Computing, <http://www.mathworks.com>.
[15] A. Cervin, M. Ohlin, D. Henriksson, Simulation of networked control systems using truetime, in: Proceedings 3rd International Workshop on Networked Control Systems: Tolerant to Faults.
[16] The Network Simulator ns-2, 2004. <http://isi.edu/nsnam/ns/>.
[17] G. Karsai, J. Sztipanovits, A. Ledeczi, T. Bapty, Model-integrated development of embedded software, Proceedings of the IEEE 91 (1) (2003) 145–164.
[18] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, P. Volgyesi, The generic modeling environment, Workshop on Intelligent Signal Processing.
[19] F. Kuhl, J. Dahmann, R. Weatherly, Creating Computer Simulation Systems: An Introduction to the High Level Architecture, Prentice Hall PTR, 1999.
[20] Omnet++ Discrete Event Simulation System, 2004. <http://www.omnetpp.org>.
[21] Modelica and Modelica Association, <http://www.modelica.org>.
[22] The Ptolemy Project, <http://ptolemy.eecs.berkeley.edu>.
[23] T. Kohtamaki, M. Pohjola, J. Brand, L. Eriksson, Piccsim toolchain – design, simulation, and automatic implementation of wireless networked control systems, in: IEEE Conference on Networking, Sensing, and Control, 2009, pp. 49–54.
[24] U. Hatnik, S. Altmann, Using modelsim, matlab/simulink and ns for simulation of distributed systems, International Conference on Parallel Computing in Electrical Engineering 0 (2004) 114–119.
[25] O. Heimlich, R. Sailer, L. Budzisz, Nmlab: a co-simulation framework for matlab and ns-2, in: 2010 Second International Conference on Advances in System Simulation (SIMUL), 2010, pp. 152 –157.
[26] A.A.-H.M. Branicky, V. Liberatore, Co-simulation tools for networked control systems, Hybrid Systems Computation and Control, Lecture Notes in Computer Science 4981 (2008) 16–29.
[27] M. Hasan, H. Yu, A. Carrington, T. Yang, Co-simulation of wireless networked control systems over mobile ad-hoc network using simulink and opnet, IET Communications 3 (8) (2009) 1297–1310.
[28] D. Riley, E. Eyisi, J. Bai, Y. Xue, X. Koutsoukos, J. Sztipanovits, Networked control system wind tunnel (NCSWT) – an evaluation tool for networked multi-agent systems, in: 4th Int. ICST Conf. on Simulation Tools and Techniques (SIMUTools), 2011.
[29] G. Hemingway, H. Neema, H. Nine, J. Sztipanovits, G. Karsai, Rapid synthesis of high-level architecture-based heterogeneous simulation: a model-based integration approach, in: Transactions of the Society for Modeling and Simulation International (SIMULATION), 2011. doi:10.1177/0037549711401950.
[30] S. Neema, T. Bapty, X. Koutsoukos, H. Neema, J. Sztipanovits, G. Karsai, Model based integration and experimentation of information fusion and c2 systems, in: 12th International Conference on Information Fusion, 2009.

[31] H. Neema, H. Nine, G. Hemingway, J. Sztipanovits, G. Karsai, Rapid synthesis of multi-model simulations for computational experiments in c2, in: Armed Forces Communications and Electronics Association – George Mason University Symposium, 2009.
[32] Gt-itm: Georgia Tech Internetwork Topology Models, 1996. <http://www.cc.gatech.edu/projects/gtitm/>.
[33] Portico RTI, 2010. <http://www.porticoproject.org>.
[34] H. LeBlanc, E. Eyisi, N. Kottenstette, X. Koutsoukos, J. Sztipanovits, A passivity-based approach to deployment in multi-agent networks, in: International Conference on Informatics in Control, Automation and Robotics (ICINCO), 2010, pp. 53–62.
[35] A. Leon-Garcia, Probability and Random Processes for Electrical Engineering, Addison-Wesley, 1993.
[36] A.T. Al-Hammouri, A comprehensive co-simulation platform for cyber-physical systems, Computer Communications (0) (2012), http://dx.doi.org/10.1016/j.comcom.2012.01.003. <http://www.sciencedirect.com/science/article/pii/S0140366412000047>.