NCCPIS: A Co-simulation Tool for Networked Control and Cyber-Physical System Evaluation

Jinzhi Lin, Ying Wu, Gongyi Wu, and Jingdong Xu

College of Information Technical Science, Nankai University
Weijin Road 94, Tianjin, 300071, China
linjinzhi@mail.nankai.edu.cn,
{wuying,wgy,xujd}@nankai.edu.cn

Abstract. As the researches on Networked Control & Cyber-Physical System (NCCPS) are growing, the requirement of reliable evaluation tools for these systems is urgent. There are several simulators, such as TureTime, Ptolemy II and so on, can be used, but they usually focus on modeling of the control dynamics, and are too simple and abstracted on the simulation of network communication. In this work, a co-simulation tool, NCCPIS is presented, which integrates the dynamic control system simulator, Ptolemy II and the network simulator, NS-2. We demonstrate the validation of the tool by presenting a case study of platoon longitudinal control in AHS (Automatic Highway System).

Keywords: Co-simulation, Ptolemy II, NS-2, Evaluation tool, NCS, CPS.

1 Introduction

In recent years, more and more researches focus on Cyber-Physical Systems (CPS), which integrate computation with the physical process. Actually, a CPS is often monitored and controlled by a Networked Control System (NCS). In this paper, we call them together as Networked Control & Cyber-Physical System (NCCPS). Basically, a NCCPS is composed of three components: physical process, computation and network [9]. The researches of NCCPS include investigating problems at the intersection of control systems, networking, and computer science [8], exploring compositional verification and testing methods [15], and so on.

Design and evaluation of NCCPS can not be accurately conducted without considering the effect of networks, it is important to develop a verification tool that not only has the ability of simulating the dynamics of the system's plant and controller, but also can simulate the realistic network environment for communication. To build a brand new tool from scratch satisfying the requirement is very difficult and unnecessary. There are several powerful model-based tools being used to simulate control systems in the academia, such as Ptolemy II [14], Simulink [11], ViSim [19], and so on. On the other hand, there are a few network simulators can exactly emulate detailed network communication, such as NS-2 [12], OpNET [4], OMNet++ [13], and so on. The feasibility of integrating a control system simulator and a network simulator to evaluate NCCPS is foreseeable.

J.J. Park et al. (Eds.): NPC 2012, LNCS 7513, pp. 85–93, 2012.

By considering the flexibility, reliability and technical supporting, we choose to integrate both open source simulators, Ptolemy II and NS-2, to build our co-simulation tool called NCCPIS. The challenges are significant, including designing the framework of NCCPIS and coordinating the synchronization of event time and data exchange in both sides of the simulators.

This paper is organized as follows. In section 2, we introduce the related work. In section 3, we present the implementation of NCCPIS. Time and data synchronization are discussed in section 4. A case study is given in section 5. Finally, this paper is concluded in section 6.

2 Related Work

As a most popular tool for validating NCS, TrueTime [7] extends Matlab/Simulink with platform related modeling concepts (i.e., network, clock and schedulers) and supports simulation of networked and embedded control systems with implementation effects [3]. However, in TureTime, the modeling of network dynamics are highly abstracted, thus it's not appropriate to evaluate the systems that require detailed low layer network communication.

For different considerations, combining different control system simulators with network simulators, some similar ideas of seeking co-simulating methods exist in a few articles. In [16], an evaluation tool called NCSWT was developed, which integrated Matlab/Simulink and NS-2 using the HLA standard for coordinating data communication and time synchronization. In [1], two approaches of extending NS-2 and one of integrating Modelica and NS-2 have been proposed. In [6], for WNCS (Wireless NCS) over MANET (Mobile ad-hoc Network), the SIMULINK-OPNET co-simulation was investigated.

3 NCCPIS Implementation

3.1 Ptolemy II and NS-2

In Ptolemy II, actors are the basic computation units that can execute concurrently and communicate through messages sent via interconnected ports [14]. There may be a special actor in a model called *Director*, which manages the execution of this model. Actors have the key flow of control methods [2, 10]:

- *setup*: Initialize the actor.
- *prefire*: Test preconditions for firing.
- *fire*: Read inputs and produce outputs.
- *postfire*: Update the state.
- *wrapup*: End execution of the actor.

The CT (continuous-time) domain in Ptolemy II aims to help the design and simulation of systems that can be modeled using ordinary differential equations (ODEs).

ContinuousDirector is the main *Director* of CT domain. It contains an *ODESolver* who's responsible to determine the integration step sizes according to ODEs or time points of interest so as to achieve an accurate simulation. Meanwhile, Ptolemy II implements a *ContinuousStepSizeController* interface to support accurate time advancement. Any actors implemented the interface can influence the passage of simulation time. The interface has following methods [10]:

- *isStepSizeAccurate*: to see if the current step is accurate.
- *suggestedStepSize*: suggests the next step size.
- *refinedStepSize*: if *isStepSizeAccurate* returns false, this method will be called, returns the step size this actor wants to refine currently.

Moreover, there is a *ContinuousStatefullComponent* interface which has a *rollBackToCommittedState* method. It's for the actors who have tentative state to roll back if the current step size is not accurate. Figure 1 shows the process of the *fire* method of *ContinuousDirector*.

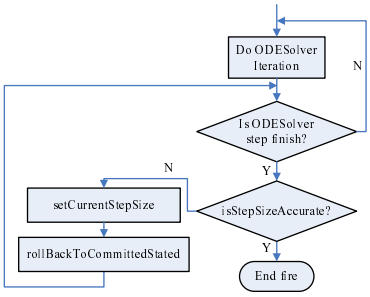

Fig. 1. The process of fire method of *ContinuousDirector*

3.2 Framework and Integration

Figure 2 shows the framework of NCCPIS. Considering supporting separated hosts simulating, we integrate the two simulators through sockets, and design a few commands for synchronization shown in table 1. On the Ptolemy II side, *NS2Node* actors are considered as the shadows of *Nodes* in NS-2, they directly participate in the simulation of Dynamic Control System. Once they have state (e.g., position and velocity) updated or packets to send, they ask *NS2Coordinator* to send the corresponding commands to NS-2 through its socket thread, and waiting for the response commands from NS-2 so as to continue the simulation. The actor *NS2Coordinator* implements *ContinuousStepSizeController* and *ContinuousStatefull-Component* interfaces intrduced previously, every time before *postfire*, *Continuous-Director* invokes *isStep-SizeAccurate*, then *NS2Coordinator* turns to consult NS-2 whether the current step size is accurate. On the NS-2 side, similarly, we develop the class *ptIIEngine* as the proxy to coordinate synchronization with Ptolemy II.

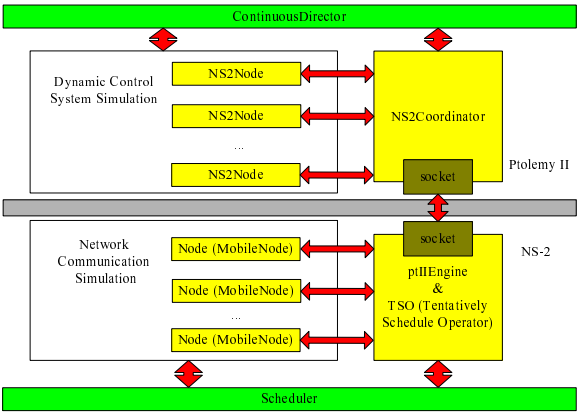

Fig. 2. The framework of NCCPIS

Table 1. Synchronization Commands

Comm	*Sender*	*Response comm*	*Description*
setdest	Ptolemy II	setdest_ok	Coordinate MobileNode to update position
broadcast	Ptolemy II	broadcast_ok	Coordinate MobileNode to broadcast packet
send_pkt	Ptolemy II	send_pkt_ok	Coordinate MobileNode to send packet
iSSA	Ptolemy II	iSSA [true/false]	Consult NS-2 if the step size is accurate
rSS	Ptolemy II	rSS [t]	Consult NS-2 the step size
sSS	Ptolemy II	sSS [t]	Consult NS-2 the next step size
rB [t]	Ptolemy II	rB_ok	Ask NS-2 to roll back to time *t*
runTo [t]	Ptolemy II	runTo_ok	Ask NS-2 to run to time *t*
recvdata	NS-2	recvdata_ok	Coordinate NS2Node to generate packet

Note: iSSA = isStepSizeAccurate, rSS = refinedStepSize, sSS = suggestedStepSize, rB = rollBackTo

4 Time and Data Synchronization

Synchronization is critical for co-simulation tools. In NCCPIS, the time when nodes send and receive a packet should be same in both simulations. For wireless network simulation, the position and velocity of nodes are essential for packet transmissions, thus, we need to synchronize the position and velocity of nodes in both simulations.

Position and velocity synchronization is simple. The position and velocity of *NS2Node* can only change in the "*Do Solver Iteration*" process shown in figure 1, as long as *NS2Node* detects the change of its state, it delegates *NS2Coordinator* to send a "*setdest*" command to NS-2. Respectively, when NS-2 receives that command, as shown in figrue 2, *ptIIEngine* inserts an *AtEvent* like "*ns at t 'nn setdest x y v*'" into scheduler, and records it in *TSO* to prepare for following up "*rollBackTo*" command. *TSO* records all events generated by the synchronization commands from Ptolemy II, if *ptIIEngine* receives a "*rollBackTo*" command, it could remove these events from the scheduler, however, if receive a "*runTo*" command, just clear them from *TSO*.

Synchronization of sending packets is similar to that of position and velocity. But, to synchronize the time of receiving packets is a little hard. As shown in figure 3, the simulation time of Ptolemy II is always ahead of NS-2, while the time NS-2 receiving packets is unpredictable, thus we can't generate a future receiving packets event for Ptolemy II in advance. As a result, the time NS-2 receiving packets is always before or equal to (if the receiving time is equal to the time Ptolemy II invokes *postfire*) the current simulation time in Ptolemy II. Obviously, we should ensure that the time NS-2 receiving packets are exactly the time Ptolemy II invokes *postfire*. To accomplish this, as shown in figure 4, we invoke *isStepSizeAccurate* to ask NS-2 to check that if there are packet receiving events in the scheduler before the current simulation time (the time Ptolemy II is going to ask NS-2 to "runTo" if all actors' *isStepSizeAccurate* return true). If existing these events, return false and return the earliest time of all events when the upcoming command "*refinedStepSize*" is invoked, else return true.

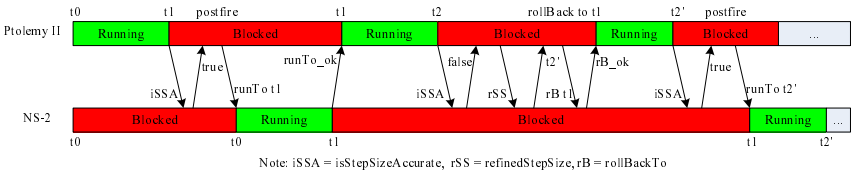

Fig. 3. The synchronization between Ptolemy II and NS-2

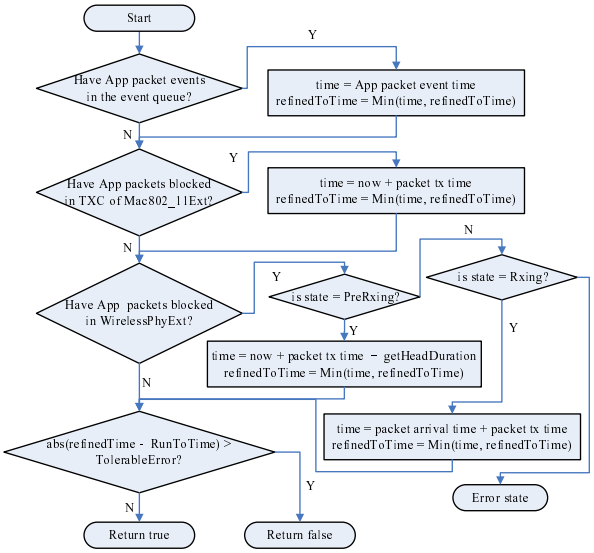

Fig. 4. The processing flow of *isStepSizeAccurate* for simulating 802.11 ad-hoc networks

Actually, to NCCPIS, only application packets matter. While invoking *isStepSize-Accurate*, we can check the event queue of the scheduler in NS-2 to see if there are packet receiving events going to happen in the Application Layer. But there is a dependency problem needs to be considered, that after some related packet events in

router, data link layer, physical layer, etc., being scheduled, packet receiving events in Application Layer may appear. By analyzing the simulation mechanism of NS-2, we found that before a packet received in application layer, it is either in the event queue of the scheduler for the upcoming scheduling, or blocked in a queue, a MAC layer, a physical layer and a channel for waiting to be put in the event queue. This means that we should check the event queue and the NS-2 components which may block the application packets for *isStepSizeAccurate* invoking. Specifically, figure 4 depicts the *isStepSizeAccurate* process flow for simulating 802.11 ad-hoc networks. There is a *TolerableError* in NCCPIS due to the *timeResolution* in Ptolemy II meaning the time precision. The default value of *timeResolution* is *1e-10*. As we can see, the time synchronization between Ptolemy II and NS-2 is within this time precision.

5 Case Study

NCCPIS in this work is intended to provide a co-simulation of dynamic control system and network communication. In this section, we present a simulation scenario to validate NCCPIS as well as show how the realistic network performs such as time-varying delays and packet losses and how they affect the overall system.

5.1 Experiment Setup

We consider a platoon of 16 vehicles running on a highway with 15 vehicles followed a lead vehicle. The lead vehicle transmits its position, velocity and acceleration measured by sensors periodically through an 802.11 wireless ad-hoc network to all the vehicles within the platoon. All vehicles are assumed to be initially traveling at the steady-state velocity of $v_0 = 17.9m/s$. Beginning at time $t = 0s$, the lead vehicle's velocity increases from its steady-state value of *17.9m/s* until it reaches its final value of *25.9m/s*: the maximum jerk and the peak acceleration values corresponding to this velocity time profiles were $2m/s^3$ and $2m/s^2$, respectively. We adopt the control law in [17] and choose the same values for the coefficients: $c_{a1} = 15$, $c_{v1} = 74$, $c_{p1} = 120$, $k_{a1} = -3.03$, $k_{v1} = -0.05$; $c_a = 5$, $c_v = 49$, $c_p = 120$, $k_a = 10$, $k_v = 25$, and the coefficients of vehicle dynamics are, for $i\ op= 1,2,\ldots$, $K_{di} = 0.3$, $\tau_i = 0.2$, $d_{mi} = 5$,$m_i = 1464$.

Figure 5 depicts the detail inside the *Wireless* component configured in Ptolemy II for each vehicle. The *NS2MobileNode* is the entity of *NS2Node* as discussed above. The *Recorder* records all the packets *NS2MobileNode* received, but outputs the currently received packet of a specific node which here is set to the lead vehicle until a new packet arrives, in fact, the *Recorder* is a *ZeroHolder*.

In NS-2, we set the network parameters consistent with IEEE 802.11p as shown in table 2, and employ the public available highway patterns and ns traffic trace generation tools presented in [5] to obtain a realistic scenario with a dynamic network topology. We consider a 6km long highway composed of 6 lanes (3 in each direction) with high traffic density, where in total 523 vehicles pass along the road in both directions with an average speed of 120km/h (33.3m/s). For this scenario, we add a 7-th lane for the platoon with the lead vehicle's initial position at 2.5km of the highway. All the normal vehicles in the highway periodically broadcast packets, thus they interfere the

communication of the platoon. We conduct three experiments with the normal vehicles broadcasting at different rates and packet sizes: (a) 500 bytes per packet, 5 packets per second; (b) 500 bytes per packet, 10 packets per second; (c) 1500 bytes per packet, 10 packets per second. The lead vehicle of the platoon broadcasts its state every 100ms by setting the interval of the *PeriodicSampler* shown in figure 5 to 0.1.

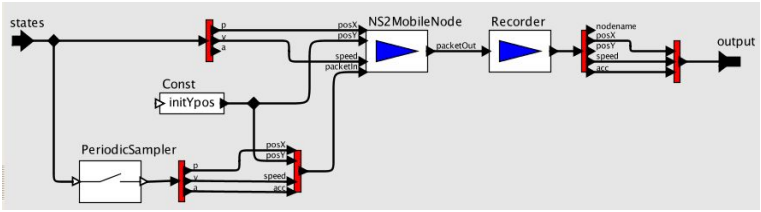

Fig. 5. The detail of the *Wireless* component configured in Ptolemy II for each vehicle

Table 2. Simulation configuration in NS-2

Category	*Parameter*	*Value*
PHY	Frequency	5.9 GHz
	Power Monitor Threshold	-102 dBm
	Transmission Power	1 mW
	SINR Preamble Capture	4 dB
	SINR Data Capture	10 dB
MAC	Slot Time	13 us
	SIFS Time	32 us
	Header Duration	40us
	Symbol Duration	8us (3 Mbps)
	Modulation Scheme	BPSK

5.2 Evaluation

Figure 6, 7 respectively shows the delays and loss of packets that the platoon vehicles received from the lead vehicle. We can see that broadcasting of non-platoon vehicles have obvious effect on communication of platoon.Figure 8 shows the co-simulation results. Compared to [17], we can infer that our co-simulation results are valid. Compared to [18], our platoon performances are better, because in our law, we used the information of the lead vehicle by adding the wireless network communication to the simulation. In addition, due to the delay and loss of packets, the performances of experiment a, b and c shown in figure 8 are worse and worse. That also proves the validation of our co-simulation. In this case study, Ptolemy II and NS-2 are both running on a RedHat virtual machine with 1GB memory on an XP host with 2GB memory and a 2.80GHz dual-core CPU. The overall 18 seconds co-simulation has taken five minutes. The communications of the 523 *MobileNodes* for the normal vehicles in NS-2 and the frequent "*rollback*" in Ptolemy II cost most all of the time.

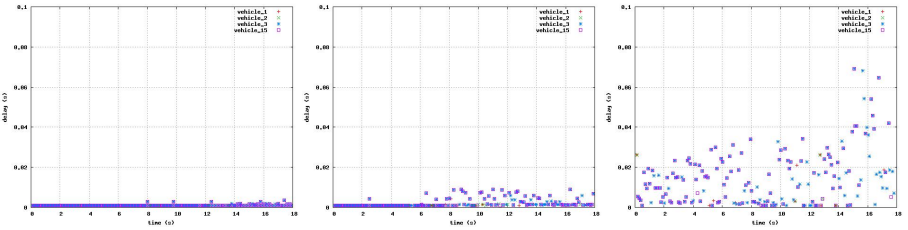

Fig. 6. The delays of packets (Experiment a, b, c, respectively)

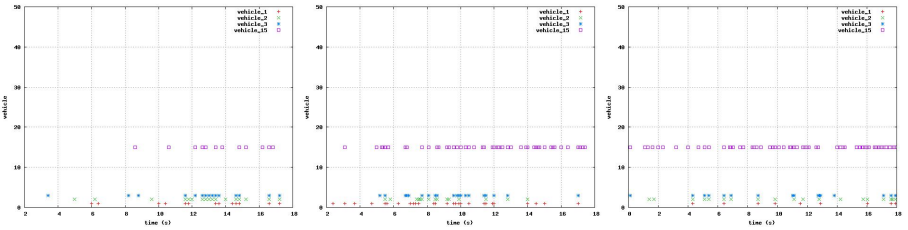

Fig. 7. The losses of packets (Experiment a, b, c, respectively)

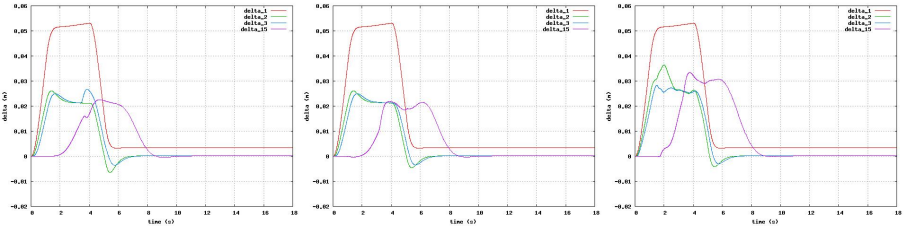

Fig. 8. The platoon performances of $\triangle_1$, $\triangle_2$, $\triangle_3$ and $\triangle_{15}$ (Experiment a, b, c, respectively)

6 Conclusion

In this work we present NCCPIS, a co-simulation tool integrating a control system simulator, Ptolemy II and a network simulator, NS-2. By studying the architectures of them, we have designed the framework of NCCPIS, developed data exchange and time synchronization mechanism. At the same time, we demonstrate the validation of the tool by presenting case studies of platoon longitudinal control in AHS. As Ptolemy II is a tool orienting multi-domains and aiming to simulate hybrid systems, we will import the co-simulation of multi-domains to NCCPIS and validate it.

Acknowledgments. This work was supported by the Research Fund for the Doctoral Program of Higher Education of China (No. 20110031110026), and the National Natural Science Foundation of China (No. 61103214).

References

1. Al-Hammouri, A.T., Branicky, M.S., Liberatore, V.: Co-simulation Tools for Networked Control Systems. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 16–29. Springer, Heidelberg (2008)
2. Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y., Zheng, H.: Heterogeneous Concurrent Modeling and Design in Java (vol. 2: Ptolemy II Software Architecture). EECS Department, University of California, Berkeley, USA (2008)
3. Cervin, A., Ohlin, M., Henriksson, D.: Simulation of Networked Control Systems using Truetime. In: 3rd International Workshop on Networked Control Systems: Tolerant to Faults, Nancy, France (2007)
4. Chang, X.: Network Simulations with Opnet. In: Proc. of Simulation Conference. Phoenix, USA (1999)
5. Chevillat, P., Jelitto, J., Truong, H.L.: Dynamic Data Rate and Transmit Power Adjustment in IEEE 802.11 Wireless LANs. International Journal of Wireless Information Networks 12(3), 123–145 (2005)
6. Hasan, M., Yu, H., Carrington, A., Yang, T.: Co-simulation of Wireless Networked Control Systems over Mobile Ad Hoc Network using Simulink and Opnet. IET Communications 3(8), 1297–1310 (2009)
7. Henriksson, D., Cervin, A., Arzen, K.E.: Truetime: Real-time Control System Simulation with Matlab/Simulink. In: Proc. of Nordic MATLAB Conference. Copenhagen, Denmark (2003)
8. Baillieul, J., Antsaklis, P.J.: Control and Communication Challenges in Networked Real-time Systems. Proc. of the IEEE 95(1), 9–28 (2007)
9. Lee, E.A.: CPS Foundations. In: 47th Design Automation Conference, Anaheim, USA, pp. 737–742 (2010)
10. Lee, E.A., Zheng, H.: Leveraging Synchronous Language Principles for Heterogeneous Modeling and Design of Embedded Systems. In: 7th ACM & IEEE International Conference on Embedded Software, New York, USA, pp. 114–123 (2007)
11. Simulink, `http://www.mathworks.com/products/simulink/`
12. The Network Simulator: ns-2, `http://www.isi.edu/nsnam/ns/`
13. Omnet++: Discrete event simulation system, `http://www.omnetpp.org`
14. Ptolemy II, `http://ptolemy.eecs.berkeley.edu/ptolemyII/index.htm`
15. Rajkumar, R., Lee, I., Sha, L., Stankovic, J.: Cyber-physical Systems: the Next Computing Revolution. In: 47th Design Automation Conference, Anaheim, USA, pp. 731–736 (2010)
16. Riley, D., Eyisi, E., Bai, J., Koutsoukos, X., Xue, Y., Sztipanovits, J.: Networked Control System Wind Tunnel (NCSWT)- An Evaluation Tool for Networked Multi-agent Systems. In: SIMUTools 2011, Barcelona, Spain, pp. 9–18 (2011)
17. Sheikholeslam, S., Desoer, C.A.: Longitudinal Control of a Platoon of Vehicles. In: American Control Conference, San Diego, USA, pp. 291–296 (1990)
18. Sheikholeslam, S., Desoer, C.A.: Longitudinal Control of a Platoon of Vehicles with no Communication of Lead Vehicle Information: A System Level Study. IEEE Transactions on Vehicular Technology 42(4), 546–554 (1993)
19. Vissim, `http://www.vissim.com/products/vissim.html`