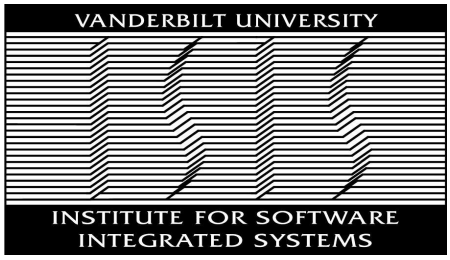


Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, 37235

PaNeCS: A Modeling Language for Passivity-based Design of Networked Control Systems

Emeka Eyisi, Joseph Porter, Joe Hall, Nicholas Kottenstette, Xenofon Koutsoukos and Janos Sztipanovits

TECHNICAL REPORT

ISIS-09-105

Abstract. The rapidly increasing use of distributed architectures in constructing real-world systems has led to the urgent need for a sound systematic approach in designing networked control systems. Communication delays and other uncertainties complicate the development of these systems. This paper describes a prototype modeling language for the design of networked control systems using passive techniques to decouple the control design from network uncertainties. The modeling language includes an integrated analysis tool to check for passivity and a code generator for simulation in MATLAB/Simulink using the TrueTime platform modeling toolbox. The resulting designs are more robust to platform effects, without costly design verification.

1 Introduction

The heterogeneous composition of computing, sensing, actuation, and communication components has enabled a modern grand vision for real-world Cyber Physical Systems (CPS). Real-world CPSs such as automotive vehicles, building automation systems, and groups of unmanned air vehicles are monitored and controlled by networked control systems (NCS). The overall system dynamics emerges from the interaction of physical dynamics and computational dynamics. The rapidly increasing use of NCS architectures in constructing these systems has lead to the urgent need for a sound systematic design. NCS systems frequently operate in safety-critical environments that require verification of the control design. Verification is complicated by heterogeneity since both physical and computational components must be taken into account. Techniques which aid in proving correctness must do so for both domains, and remain compatible with established engineering processes [1].

Model-based design for embedded control systems involves creating models and checking correctness at different stages in the development process [2]. Model-based design flow progresses along precisely defined abstraction layers, typically starting with control design. Then control design models are passed onto the system-level design stage for the specification of platform details, code organization, and deployment details. The final stage involves integration and testing on the deployed system.

This design approach cannot be applied to NCS because heterogeneity creates a number of challenges that include ensuring controller stability and performance for physical systems in the presence of network uncertainties (e.g. time delay, packet loss). Further, downstream code modifications during testing and debugging invalidate results from earlier design-time analysis and any component change often results in "restarting" the design process. These are challenges that stem from the coupling between design concerns in different domains. For example, separate specification of controller requirements and platform designs seems natural. However, correct controller function depends on the details of both. Similar coupling exists between different stages of the development process, for example between functional design models and software design models.

A number of research projects seek to address the problems of model-based design for NCS. The ESMoL modeling language for designing and deploying time-triggered systems shares many structural design concepts [3]. The ESMoL tools include schedule determination for time-triggered communications, code generation, and a portable time-triggered virtual machine. AADL [4] is a textual language and standard for specifying deployments of control system designs in data networks [5]. AADL projects also include integration with verification and scheduling analysis tools. The Metropolis modeling framework [6] aims to give designers tools to create verifiable system models. Metropolis integrates with SystemC, the SPIN model-checking tool, and other tools for scheduling and timing analysis.

In order to tackle the challenges of developing software for NCS, we propose an automated model-based approach on top of passivity. We use Model-Integrated Computing [2] to develop a domain specific modeling language (DSML) called the Passive Network Control Systems language (PaNeCS) based on the passive control architecture presented in [7]. PaNeCS raises the level of abstraction of networked control system design, and uses passivity to decouple the control design from network uncertainties. As a result, we can analyze component correctness properties independently, and then establish global correctness from the structure of their interconnections. This stands in contrast to global techniques that analyze all details of all parts of a model simultaneously.

The paper is organized as follows: Section 2 presents design challenges in the compositional design of NCS. Section 3 presents passivity-based control of NCS. Section 4 presents our prototype modeling language. Section 5 presents an analysis tool for checking passivity. Section 6 presents a model interpreter for automatically generating Matlab/Simulink simulation code using the TrueTime platform modeling toolbox. Section 7 shows a case study of a NCS consisting of two discrete plants and a controller. Section 8 provides our conclusion.

2 Compositional Design of Networked Control Systems

Building systems from components is central to all engineering disciplines to manage complexity, decrease time-to-market, and contain cost. The feasibility of component-based design depends on two key conditions: *compositionality* meaning that system-level properties can be computed from local properties of components and *composability* meaning that component properties do not change as they interact with other components. Lack of compositionality and composability leads to behavioral properties that can be verified or measured only by system-level analysis (and/or testing), which can be inefficient for complex real-world systems.

NCS involve the interaction of physical dynamics, computational dynamics, and communication networks. This heterogeneity does not go well with current methods of compositional design for several reasons. The most important principle used in achieving compositionality is separation of concerns (in other words, defining design viewpoints). Separation of concerns works if the design views are

orthogonal, i.e. design decisions in one view do not influence design decisions in other views. Unfortunately, achieving compositionality for multiple physical and functional properties simultaneously is a very hard problem because of the lack of orthogonality among the design views.

Control designers create models for both physical systems and controllers using tools like Simulink and Stateflow [8]. Models generally consist of functional data flow networks and variants of state machines. Deployment of a control design such as a Simulink design to a networked architecture introduces uncertainties due to time-varying delay, data rate limitations, jitter, and packet loss. This invalidates many of the simulation results for the design. Deployment of the design to the physical environment is often expensive, and failure during testing can be costly as well. An increasingly accepted way to address the problems is to enrich abstractions in each layer with implementation concepts. An excellent example for this approach is TrueTime [9] that extends Matlab/Simulink with platform-related modeling concepts (networks, clocks, schedulers) and supports simulation of networked and embedded control systems with the modeled implementation effects. While this is a major step in improving designers understanding of implementation effects, it does not help in decoupling design layers and improving orthogonality across design concerns. A control designer can now factor in implementation effects (e.g., network delays), but still, if the implementation changes, the controller may need to be redesigned.

Control systems are often verified using complex optimization techniques. For example, linear matrix inequalities (LMIs) can model many important controller properties (e.g. stability, response time, reachability). In a system built from the composition of multiple blocks, such analysis quickly becomes intractable. In order to assess global stability, designers must build a single, large model which includes all possible states of the system. In contrast, passivity theory can ensure global stability (in a robust way) by a combination of component analysis and specific rules for composition of passive components.

Downstream code modifications during testing and debugging may invalidate models and results from earlier design-time analyses. Changes made during design, development, and testing cycles may cause extensive software revisions and force expensive re-verification. Model-integrated computing tools provide automated software generation, analysis, and system configuration directly from models [2]. PaNeCS supports forward generation of platform-specific simulation models as well as lightweight passivity analysis. Code generation and other verification tools are planned for future versions.

3 Passivity-Based Control of Networked Control Systems

Our approach for designing NCS is based on passivity. There are various precise mathematical definitions for passive systems [10]. Essentially all definitions state that the output energy must be bounded so that the system does not produce more energy than was initially stored. Passive systems have a unique property that when connected in either a parallel or negative feedback manner the overall

system remains passive. Passivity provides an inherent safety – passive systems are insensitive to implementation uncertainties, so passivity can be exploited in the design of NCS. The main idea is that by imposing passivity constraints on the component dynamics, the design becomes insensitive to network effects, thus establishing orthogonality (with respect to network effects) across the various design layers. This separation of concern allows the model-based design process to be extended to networked control systems.

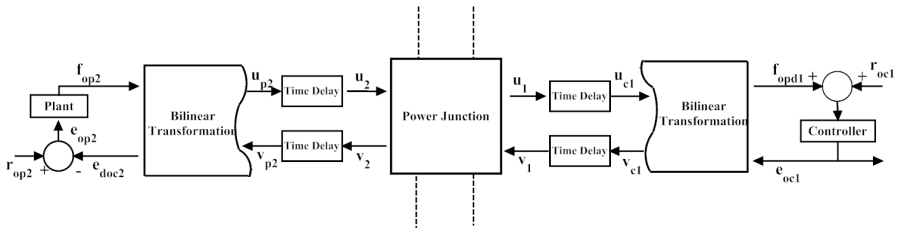


Fig. 1. A networked control system

We briefly discuss the passivity based control architecture for multiple plants controlled by a single controller via a network [7]. In Fig. 1, the Bilinear Transform block represents the transformation between signals and wave variables. Wave variables were introduced by Fettweis in order to circumvent the problem of delay-free loops and guarantee a realizable implementation for wave digital filters [11]. Wave variables allow systems to remain passive while transmitted data over a network subject to arbitrary fixed time delays and data dropouts [12], [13]. A detailed mathematical description for wave variables in NCS can be found in [14]. In Fig. 1, $u_{pk}(i)$ (k=1,2), can be thought of as a sensor output data in wave variable form for each plant. Likewise, $v_{cj}(i)$ (where j=1,2) can be thought of as a command output in wave variable form for the controller.

The power junction in Fig. 1 is an abstraction used to interconnect wave variables from multiple controllers and multiple plants in parallel such that the total power input is always greater than or equal to the total power output. It provides a formal way to construct a networked control system [7]. The power junction makes it possible for a single controller to control multiple plants over a network such that the overall system remain stable. A more detailed mathematical definition of the power junction can be found in [7]. In Fig. 1, the power junction has waves entering and leaving as indicated by the arrows. The waves entering the power junction from the controller are the delayed version of the waves leaving the controller, as indicated by the time delay block. Also, the waves entering the controller are the delayed version of the waves leaving the power junction. Likewise, the waves entering the plant are the delayed version of the waves leaving the power junction and the waves entering the power junction are the delayed version of the waves leaving the plant.

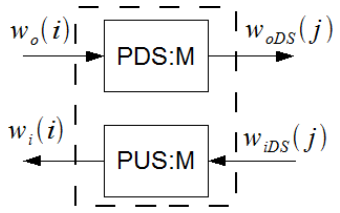


Fig. 2. The passive upsampler and passive downsampler.

In the design of NCS there is a need to reduce digital control traffic in the network. In order to achieve this we use the passive upsampler (PUS) and passive downsampler (PDS) pair. The Fig. 2 represents the passive upsampler (PUS) and passive downsampler (PDS) construction. $w_o(i)$ denotes a discrete wave variable going out of a wave transform block. For example, in Fig. 1, $v_{c1}(i) and\ u_{p2}(i)$, the wave variables going out of the Bilinear Transformation block, are unique $w_o(i)$'s. Similarly, $w_i(i)$ represents the respective discrete wave variable going into a wave transform block. For example, in Fig. 1 $u_{c1}(i) and v_{p2}(i)$, the wave variables going into the Bilinear Transformation block, are unique $w_i(i)$'s. Essentially, the PUS and PDS provide the upsampled and downsampled versions of their wave variable inputs respectively while preserving passivity. Hence, by preserving passivity the stability of the system is maintained.

4 PaNeCs

We introduce the passivity-based modeling language (PaNeCS). The modeling language is developed using a meta-configurable tool, the Generic Modeling Environment (GME), from the Model Integrated Computing (MIC) tool suite [15].

4.1 Components

The language top level consists of four main components: the **PlantSystem**, the **ControllerSystem**, the **PowerJunction** and the **WirelessNetwork**.

PlantSystem Fig. 3 shows a part of the metamodel that describes the plant subsystem. *Plant* represents a model for any discrete linear time-invariant (LTI) system and can be extended to a nonlinear system. The dynamics of the *Plant* are represented in the following state space form:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k). \end{aligned} \tag{1}$$

The *Plant* dynamics are parameterized by matrix attributes A, B, C, D, and a scalar *SamplingTime*. The attributes can be specified using any valid expression that evaluates to the proper dimensions. *BilinearTransformP* represents a model for the wave scattering technique for transforming the wave variables received

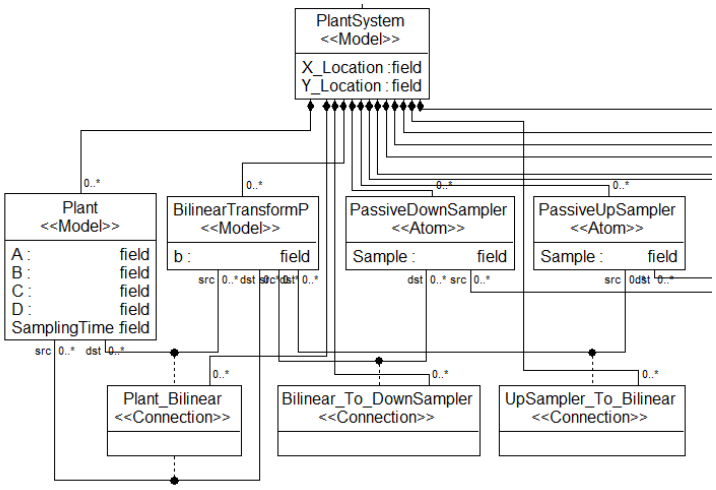


Fig. 3. PlantSystem Language

from the power junction into control input to the plant and for transforming the plant output signal into wave variables that are transmitted over the network. *PassiveUpSampler and PassiveDownSampler* pair represent components that reduce digital control traffic while maintaining system stability.

ControllerSystem Fig. 4 shows the part of the language that describes the controller subsystem. *DigitalController* is a model representing the algorithm for controlling the networked plants. Similar to the model of the *Plant* in the **PlantSystem**, the *DigitalController* is modeled as a LTI system and its dynamics can be represented in the state space form of Eq. (1). Therefore, the *DigitalController* parameters have similar attributes to the *Plant*. *BilinearTransformC* is similar to the *BilinearTransformP* described in the **PlantSystem**. *ZeroOrderHold* represents a component that holds its input for the time period specified in the sampling time attribute. *ReferenceInput* represents the desired signal to be tracked by the plants.

Power Junction Fig. 5 shows the part of the language that describes the power junction. The PowerJunction can contain ports for the connection of the plants and controllers. They are briefly described as follows: *PowerInputPowerOutput* represents a port through which the **PlantSystem** connects to the **PowerJunction**. Using the PowerInputPowerOutput entity, the **PowerJunction** sends calculated control signals to the **PlantSystem** and also receives sensor signals from the **PlantSystem**. *PowerOutputPowerInput* represents a port through which the **ControllerSystem** can connect to the **PowerJunction**. Using the PowerOutputPowerInput entity, the **PowerJunction** sends the averaged sensor signal to the **ControllerSystem** and receives the calculated control signal from the **ControllerSystem**.

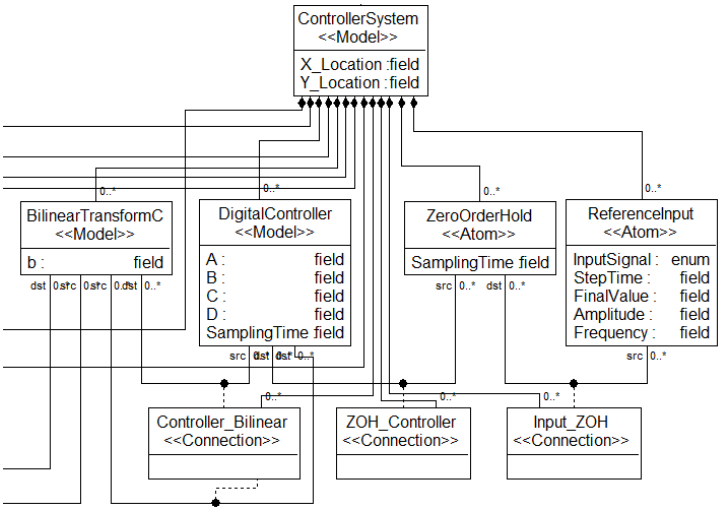


Fig. 4. ControllerSystem Language

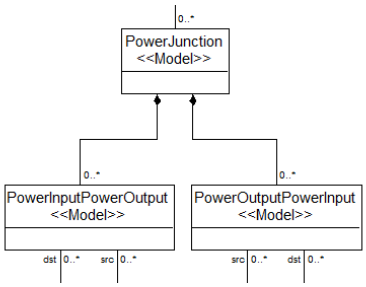


Fig. 5. PowerJunction Metamodel

WirelessNetwork Fig. 6 represents the network and its parameters for the NCS. The **WirelessNetwork** model provides modifiable parameters for simulation. *Data rate* sets the throughput for simulating network activity. *DisturbancePacketSize* configures the size of simulated disturbance attack packets on the network (introduces delays). This provides a way for simulating the NCS under non-optimal conditions. *DisturbancePeriod* configures the frequency of disturbance attacks on the network.

4.2 Aspects

Our modeling language has two aspects: **Control Design Aspect** and **Platform Aspect**. The **Control Design Aspect** visualizes the controller modeling layer. This includes the plants, controller, and power junction, as well as their interconnections – indicating the flow of control and sensor signals.

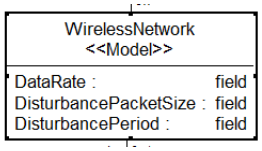


Fig. 6. Wireless Network Model

The **Platform Aspect** visualizes the physical platform layer. This model view shows the physical components of the NCS. The entities in this view include the plants, controller, and the wireless network as well as their interconnections indicating the flow of data packets over the network. Though the plants and controller appear in both aspects, in the Platform aspect they represent physical entities rather than control design concepts.

4.3 Structural Semantics

The language semantics require structural constraints that cannot be captured with the metamodeling notations described in the sub-languages described above. Using the Object Constraint Language (OCL), we can describe well-formed rules for models, enabling "correctness-by-construction" for passive designs.

After the instance model is created, the constraint checker interprets the defined constraints and reports any violations. In order to conform to the passive control architecture, only a single connection is allowed from a Plant to a BilinearTransformP block. This constraint can be specified using OCL and a violation of this constraint will alert and notify the designer to correct the design error. Hence, OCL helps in defining precise control of static semantics of the language. Three classes of constraints were implemented Cardinality Constraints, Connection Constraints and Unique Name Constraints. *Cardinality Constraints* ensure that the required and the correct number of components are used in the NCS design. For example, for each **PlantSystem** model there must be one *Plant*. *Connection Constraints* restrict the number of allowable connections between components. For example, in the **PlantSystem** model there can be only one connection going from *Plant* to *BilinearTransformP*. *Unique Name Constraints* ensure the uniqueness of the names of components in the Plant and Controller subsystems as well as in the top level model of the NCS. For example, in the **PlantSystem** no two components can have the same name.

An example of the OCL constraint implementation is shown below. This specifies that the number of allowable connections from a BilinearTransformC model to a DigitalController to be one.

```
Description: There must be only one connection from
 BilinearTransformC to the DigitalController

Equation: let dstCount = ...
 self.attachingConnections("src",Controller_Bilinear)->size in
```

```
dstCount <> 0 implies dstCount = 1
```

5 Passivity Analysis

In order to achieve the desirable properties observed in passive systems that ensure the designed networked control systems is insensitive to network effects, we have to analyze the components of the networked control system and make sure they satisfy the imposed passivity constraints.

The power junction element enforces some simple mathematical constraints which ensure passivity for interconnected components. Further, the component interconnections are restricted in such a way that they are "correct-by-construction". Only valid (parallel) connections are allowed to the power junction, so any interconnected system of passive components in the language will be globally passive. The modeling language and its constraints encode the passive composition semantics, greatly reducing the analysis burden for determining passivity (and hence stability [7], [10], [14]) of the composed system design.

Due to the "correct-by-construction" approach we use in designing networked control, we only analyze the *Plant* and *DigitalController* for passivity. If the *Plant* and *DigitalController* both satisfy the passivity constraints, the network control system as whole also satisfies the passivity principles.

The dynamics of the *Plant* and *DigitalController* models can each be defined by Eq.(1) and are characterized by the matrices A, B, C, D of size compatible with the number of inputs and outputs in the system and the number of states in the model. The passivity constraints for these models is defined by Linear Matrix Inequality (LMI) constraints [16]. For example, the LMI formula for strict output passivity for an LTI digital controller is given by

$$\begin{bmatrix} A^TPA - P - \hat{Q} & A^TPB - \hat{S} \\ (A^TPB - S)^T & -\hat{R} + B^TPB \end{bmatrix} \leqslant 0$$

$$\begin{aligned} \hat{Q} &= C^TQC, \quad \hat{S} = C^TS + C^TQD \\ \hat{R} &= D^TQD + (D^TS + S^TD) + R \end{aligned} \tag{2}$$

$$\exists \varepsilon > 0,\ Q = -\varepsilon I,\ R = 0,\ S = \frac{1}{2}I$$

The CVX semidefinite programming (SDP) tool is used in a Matlab script for solving the LMI.

The analysis of the *Plant* and *DigitalController* components of the networked control system for passivity is done automatically by an integrated Matlab analysis function. Each component is assumed to have a linear time-invariant (LTI) discrete-time model, so we use LMIs together with the `CVX` semidefinite programming tools for Matlab [17, 18]. On invocation (i.e. the modeler presses a button),

a C++ model interpreter within GME [15] visits each component, and invokes the analysis function. Any components failing the passivity test are reported to the user.

6 Code Generation

The main objective of the code generator is to generate MATLAB code that maps the models generated using the modeling language to Simulink models that represent the networked control system.

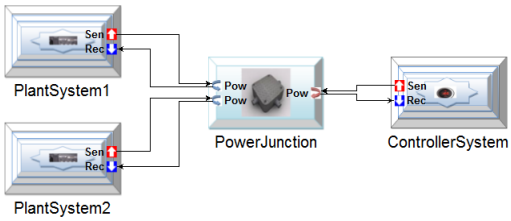
We develop a model interpreter that is used to synthesize simulation code from an instance model of the passivity based modeling language. The interpreter is developed in C++ using the Builder Object Network (BON2) API provided with GME [15]. The interpreter traverses all the entities of a particular networked control system instance model and extracts model parameters. These parameters and model structure are used to generate MATLAB files for configuring and building Simulink and TrueTime models to simulate the NCS. TrueTime is a simulation environment that extends MATLAB/Simulink with implementation-related modeling concepts such as networks, clocks and schedulers [9] that is well suited for the simulation of networked control systems.

The model interpreter creates translation rules between models and desired outputs. The entities in the instance model each map to a set of equivalently-defined components in Simulink and components from an advanced Simulink passivity-based control library. For example, the Plant and DigitalController entities each map to an equivalent discrete state-space Simulink block. For these two entities the parameters for the equivalent Simulink blocks are instantiated using the parameter values entered by the user describing the dynamics of the entities. These parameters include the A, B, C and D matrices as well as the sampling time. For an example using the passivity control library, the BilinearTransformP in the PlantSystem maps to an equivalent wave variable transform block in our library.

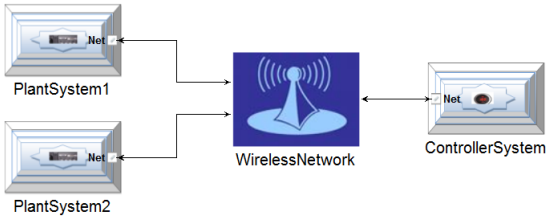
The WirelessNetwork entity maps to an equivalent TrueTime wireless network block used to simulate the network dynamics. Also, each of the PlantSystems and ControllerSystems map to Truetime Kernels, which provide interfaces for receiving and sending data over the TrueTime wireless network as well as processing and sending data to components of the subsystem.

7 Case Study

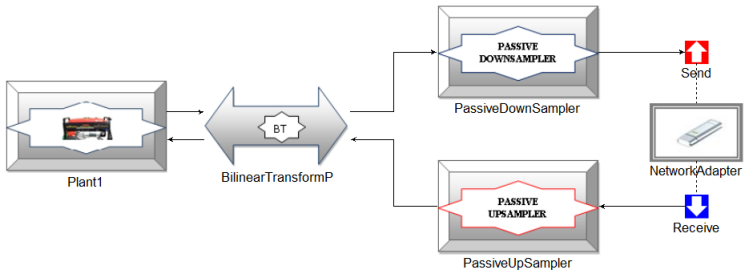
We introduce a case study to demonstrate our design approach and also show that networked control systems designed using this approach are robust and remain stable when subject to uncertain network effects. We create a networked control system which involves the control of two discrete plants using a single controller. The controller controls the two discrete plants to track a specified reference signal. The goal of the experiment is to model the network control system and generate a simulation of the behavior of the system. Fig. 7a and



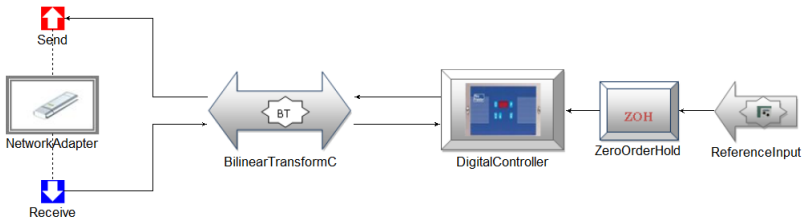
(a) Control Design Aspect



(b) PlatformAspect



(c) Plant Subsystem



(d) Controller Subsystem

Fig. 7. Sample Model of a Networked Control System

7b respectively show the control design and platform aspects of the instance model respectively. Also, Fig. 7c shows the details of the plant system while

Fig. 7d show the details of the controller system. The two plants modeled in the experiment were simple integrators (with masses of 2kg and .25kg respectively) which are discretized. The plants' dynamics were modeled in state space form and the corresponding A, B, C and D matrices as well as the samplint time, T_s were provided as parameters to the instance model. We used a proportional controller as the digital controller. This controller was used to control the plants to track a user-specified reference. The digital controller was also modeled in state space form and the A, B, C and D matrices and also the sampling time, T_s were provided as input parameters to the instance model. The parameters for the dynamics of the plants and controller is provided in Table 1. The analysis tool checks and verifies that the *Plant* and *DigitalController* models satisfy the passive constraints. If the passivity constraints are satisfied, the code generator is used to generate code for creating a platform-specific Simulink simulation model from the parameters and design models in the modeling language.

PaNeCS provides the flexibility to easily model networked control systems using passivity and more quickly configure the model parameters of the system for many different adaptations when compared to a "manual approach". Using PaNeCs we tested the dynamics of the NCS by running different experiments under different network conditions by adjusting parameters in the language and then generating code for simulating each configuration of the model. The parameters for the simulations are provided in Table 2.

Experiment 1: Nominal Conditions In experiment 1, the system operates without the introduction of disturbance attacks. The three data rates considered are 0.1s, 0.5s and 1s. The data rates were achieved by modifying the *Sample, M* parameters of the PassiveUpSampler and PassiveDownSampler entities. We only present the results of the NCS for the data rate of 0.1s. Fig. 8 displays the velocity of the plants and the reference velocity provided to the controller. The plants closely track the reference velocity. The round trip delay for each plant, has very little effect on the stability of the plants' velocity response. The delay can be attributed to the internal processing of the plants and controllers rather than network delay itself.

Table 1. Plant and Controller Dynamics.

	A	B	C	D	T_s
Plant1	1	1	.005	.0025	$.01s$
Plant2	.996	1	.04	.02	$.01s$
Controller	0	0	0	10π	$.1s$

Experiment 2: Network disturbances In experiment 2, a disturbance attack is introduced in the network. A disturbance node is configured using the *DisturbancePeriod* and DisturbancePacketSize from the **WirelessNetwork** model.

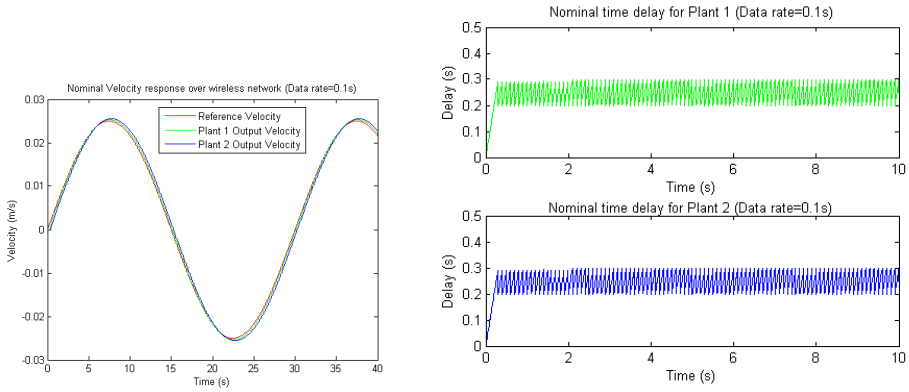


Fig. 8. Nominal velocity response and time delays (Data rate=0.1s)

Packets are sent over the network based on the value of a uniformly generated random number. Similar to Experiment 1, three cases based on the data rates are tested but we only present the results for the data rate of 0.1s. Fig. 9 shows the velocity response of the plants and the time delay for each plant. The results show that even in the presence of disturbance attacks, the plants remain stable in tracking the reference velocity. This demonstrates the advantage of the passivity approach we use in designing networked control systems which guarantees the stability of the NCS in the presence of uncertainties due to network effects.

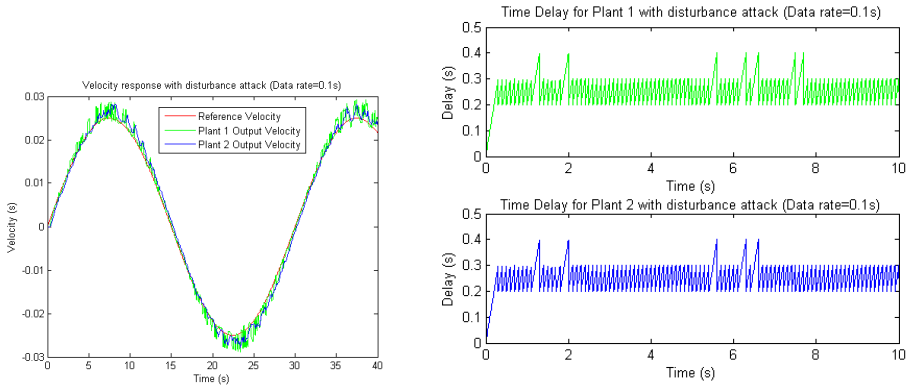


Fig. 9. Velocity response and time delays with disturbance attack (Data rate=0.1s)

Table 2. Simulation Parameters Summary.

	Data Rates		
	$0.01s$	$0.05s$	$0.1s$
Plant1,M	10	50	100
Plant2,M	10	50	100
Disturbance $T_s = 0.01$ $Packetsize = 110,000bits$			

8 Conclusion and Future Work

Our model-based approach simplifies the process of designing networked control system. We presented PaNeCS, a modeling language that is used in designing passivity-based networked control systems. We have presented an analysis tool that is used in testing system components for passivity. We have also discussed the model interpreters that generate code for simulation in MATLAB/Simulink using the TrueTime platform modeling toolbox. A case study involving the control of multiple discrete plants over a wirless network was used to demonstrate the details of models generated using the modeling language as well as the resulting simulation of the generated networked control system. The results showed the networked control system designed using our approach is robust and insensitive to uncertainties due to network effects. Our future work focuses on two major directions: (i) extending the language to include nonlinear and more complex systems,(ii) generating executables for deployment on actual systems.

References

1. Henzinger, T., Sifakis, J.: The embedded systems design challenge. In: FM: Formal Methods. Lecture Notes in Computer Science 4085. Springer (2006) 1–15
2. Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: Model-integrated development of embedded software. Proceedings of the IEEE **91**(1) (Jan. 2003)
3. Porter, J., Karsai, G., Volgyesi, P., Nine, H., Humke, P., Hemingway, G., Thibodeaux, R., Sztipanovits, J.: Towards model-based integration of tools and techniques for embedded control system design, verification, and implementation. In: Workshops and Symposia at MoDELS 2008, Springer LNCS 5421, Toulouse, France (2008)
4. AS-2 Embedded Computing Systems Committee: Architecture analysis and design language (aadl). Technical Report AS5506, Society of Automotive Engineers (November 2004)
5. Hudak J. and Feiler P.: Developing aadl models for control systems: A practitioner's guide. Technical Report CMU/SEI-2007-TR-014, CMU Software Engineering Institute (SEI) (2007)
6. Balarin, F., Watanabe, Y., Hsieh, H., Lavagno, L., Paserone, C., Sangiovanni-Vincentelli, A.L.: Metropolis: an integrated electronic system design environment. IEEE Computer **36**(4) (April 2003)
7. Kottenstette, N., Hall, J., Koutsoukos, X., Antsaklis, P., Sztipanovits, J.: Digital control of multiple discrete passive plants over networks. Technical report, Institute for Software Integrated Systems, Vanderbilt University (March 2009 Submitted)

8. The MathWorks, Inc.: Simulink/Stateflow Tools. http://www.mathworks.com
9. Ohlin, M., Henriksson, D., Cervin, A.: TrueTime 1.5 Reference Manual. Dept. of Automatic Control, Lund University, Sweden. (January 2007) http://www.control.lth.se/truetime/.
10. Kottenstette, N., Antsaklis, P.J.: Stable digital control networks for continuous passive plants subject to delays and data dropouts. In: Proceedings of the 46th IEEE Conference on Decision and Control. (2007) 4433 – 4440
11. Fettweis, A.: Wave digital filters: theory and practice. Proceedings of the IEEE **74**(2) (1986) 270 – 327
12. Secchi, C., Stramigioli, S., Fantuzzi, C.: Digital passive geometric telemanipulation. In: IEEE International Conference on Robotics and Automation. (2003) 3290 – 3295
13. Berestesky, P., Chopra, N., Spong, M.W.: Discrete time passivity in bilateral teleoperation over the internet. In: IEEE International Conference on Robotics and Automation. (2004) 4557 – 4564
14. Kottenstette, N., Koutsoukos, X., Hall, J., Antsaklis, P.J., Sztipanovits, J.: Passivity-based design of wireless networked control systems for robustness to time-varying delays. Real-Time Systems Symposium, (RTSS 2008) (December 2008) 15–24
15. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., IV, C.T., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The generic modeling environment. Workshop on Intelligent Signal Processing (May 2001)
16. Kottenstette, N., Antsaklis, P.J.: Time domain and frequency domain conditions for passivity. Technical Report ISIS-2008-002, Institute for Software Integrated Systems, Vanderbilt University and University of Notre Dame (November 2008)
17. Grant, M., Boyd, S.: Cvx: Matlab software for disciplined convex programming. http://stanford.edu/ boyd/cvx (February 2009)
18. Grant, M., Boyd, S.: Graph implementations for nonsmooth convex programs. Recent Advances in Learning and Control (a tribute to M. Vidyasagar), Springer Lecture Notes in Control and Information Sciences (2008) 95–110