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

Institute for Software Integrated Systems
Vanderbilt University
2015 Terrace Place, Nashville, TN 37203 USA
emeka.p.eyisi@vanderbilt.edu

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 passivity 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 by construction more robust to platform effects and implementation uncertainties.

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). NCS involve the interaction of physical dynamics, computational dynamics, and communication networks. This heterogeneity does not go well with current methods of compositional design. The most important principle used in achieving compositionality is separation of concerns which 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.

Model-based design for embedded control systems involves creating models and checking correctness at different stages in the development process [1]. Model-based design flow progresses along precisely defined abstraction layers, typically starting with control design followed by system-level design for the

specification of platform details, code organization, and deployment details, and the final stage of integration and testing on the deployed system. This design approach cannot be applied directly to NCS because domain heterogeneity and tight coupling between design concerns create a number of challenges. Ensuring controller stability and performance for physical systems in the presence of network uncertainties (e.g. time delay, packet loss) couples the control and system-level design layers. In addition, 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.

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 control systems explicitly captures in model structure many of the essential relationships in an embedded design[2]. The ESMoL tools include schedule determination for time-triggered communications, code generation, and a portable time-triggered virtual machine. AADL [3] is a textual language and standard for specifying deployments of control system designs in data networks [4]. AADL projects also include integration with verification and scheduling analysis tools. The Metropolis modeling framework [5] 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 designing NCS, we propose an automated model-based approach based on passivity control theory. We used Model-Integrated Computing [1] to develop a domain specific modeling language (DSML) called the Passive Network Control Systems language (PaNeCS). Our approach is based on the passive control architecture presented in [6] which provides the theoretical foundations for analysis and design of NCS emphasizing robustness to network delays and packet loss. This paper focuses on the design of the DSML as well as a compositional tool for passivity analysis and code generation for Matlab/Simulink/Truetime models. We aim to address a number of significant challenges:

- 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. PaNeCS supports forward generation of platform-specific simulation models as well as passivity analysis of system components.
- 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 would have to build a single, large analysis model which includes all possible state variables in the system. In contrast, the passive control architecture can ensure global stability (in a robust way) by a combination of component analysis and specific rules for composition of passive components.

– Control designers create models for both physical systems and controllers using tools like Simulink and Stateflow [7]. Deployment of a control design such as a Simulink model to a networked architecture introduces uncertainties due to time-varying delay, data rate limitations, jitter, and packet loss. Deployment of the design is often expensive, and failure during testing can be costly. An increasingly accepted way to address these problems is to enrich abstractions in each layer with implementation concepts. An excellent example for this approach is TrueTime [8] that extends Simulink with platform-related modeling concepts (i.e., 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 understanding of implementation effects, it does not help in decoupling design layers and improving orthogonality across design concerns. A control designer can factor in implementation effects (e.g., network delays), but if the implementation changes the controller may need to be redesigned. Our approach imposes passivity constraints on the component dynamics, so that the design becomes insensitive to network effects, thus establishing orthogonality (with respect to network effects) across the controller design and implementation design layers.

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

2 Passivity-Based Control of Networked Control Systems

Our approach for designing NCS is based on passivity theory. There are various precise mathematical definitions for passive systems [9]. 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 certain implementation uncertainties [10] [11][12], 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 concerns allows the model-based design process to be extended to networked control systems which is what our model-based approach provides.

We briefly discuss the passivity based control architecture for multiple plants controlled by a single controller via a network [6]. Fig. 1 depicts a sample networked control system where only one plant is shown. The Bilinear Transform

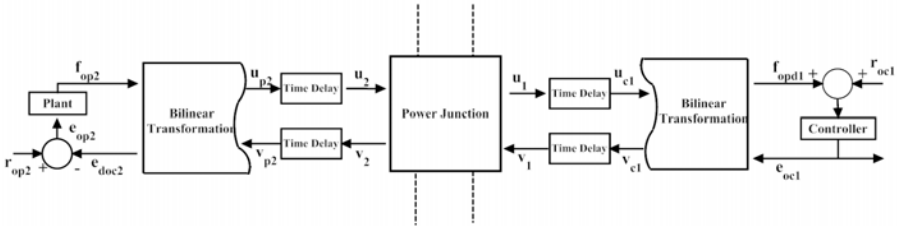


Fig. 1. A networked control system

block represents a 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 digital filters [10]. Wave variables also allow systems to remain passive while transmitted data over a network subject to arbitrary fixed time delays and data dropouts [11], [12]. In Fig. 1, $u_{pk}(i)$ (k=1,2), can be thought of as sensor output data in wave variable form from each plant. Likewise, $v_{cj}(i)$ (where j=1,2) can be thought of as a command output in wave variable form from 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 input power is always greater than or equal to the total output power. This provides a formal way to construct a networked control system. The power junction makes it possible for a single controller to control multiple plants over a network and guarantee that the overall system remains stable. A detailed mathematical definition of the power junction can be found in [6]. 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 network-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 waves leaving the power junction and the waves entering the power junction are the delayed version of waves leaving the plant.

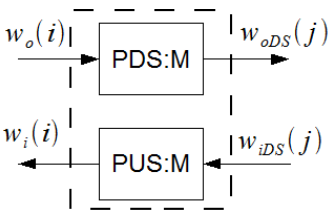


Fig. 2. The passive upsampler and passive downsampler.

Due to bandwidth constraints, the controller typically runs at a slower rate than the sensors and actuators of the plants. In order to preserve passivity in the multi-rate digital control network we use the passive upsampler (PUS) and pas-

sive downsampler (PDS) pair to handle the data rate transitions. Fig. 2 depicts the passive upsampler (PUS) and passive downsampler (PDS). $w_o(i)$ 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 each connected as $w_o(i)$ to their respective downsampler blocks. Similarly, $w_i(i)$ represents the respective discrete wave variable going into a wave transform block. $u_{c1}(i)$ and $v_{p2}(i)$, correspond to the $w_i(i)$ connections in Fig. 2. The PUS and PDS provide the upsampled and downsampled versions of their respective wave variable inputs while preserving passivity. The block parameter M is the sampling ratio – the data rate of the fast side of the connection divided by the data rate on the slow side.

3 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 [13]. GME provides a metamodeling environment similar to UML. The class stereotypes are defined as follows: Models are entities that may contain other objects while Atoms are indivisible entities which cannot contain other objects; Connections are association classes used to describe the relationship between two entities. It represents a line that connects two entities of a model. Connectors signified by "**.**" specify a visualization for a connection in the model. Associations to the connector have possible roles ("src" and "dst") to define the allowed direction of a connector.

3.1 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 PaNeCS 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 by the following state space equations:

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

The *Plant* dynamics are parameterized by matrix attributes A, B, C, D, and a scalar *SamplingTime*. The attributes can be specified using any valid Matlab expression that evaluates to the proper dimensions. *BilinearTransformP* represents a model for the wave scattering technique for transforming the wave variables received 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 the PUS and PDS pair discussed in Section 2.

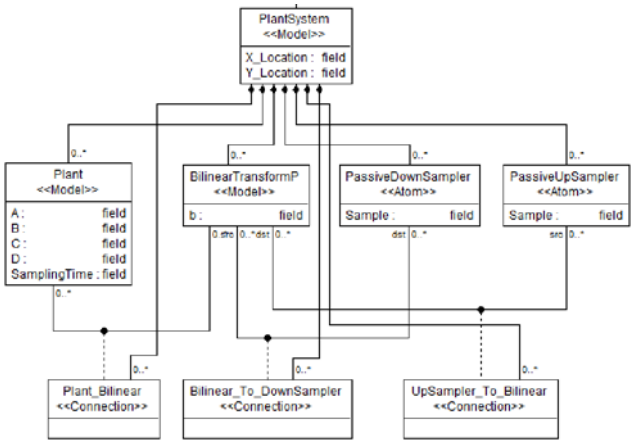


Fig. 3. PlantSystem portion of the Metamodel

ControllerSystem Fig. 4 shows the part of the language that describes the controller subsystem. *DigitalController* is a model representing the algorithm

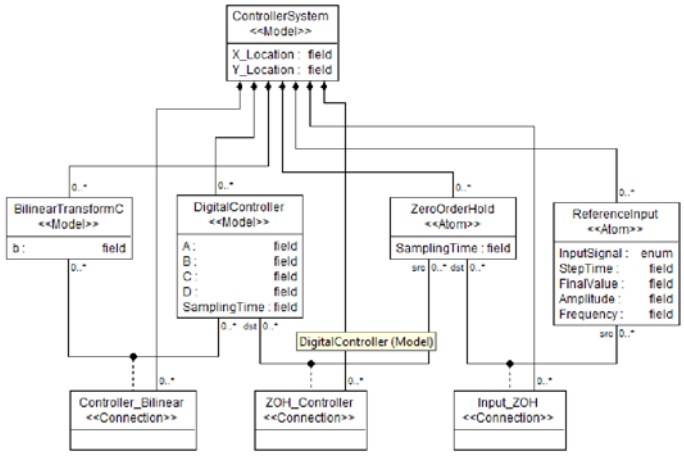


Fig. 4. ControllerSystem portion of the Metamodel

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 also 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

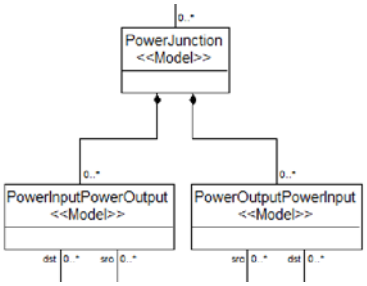


Fig. 5. PowerJunction portion of the Metamodel

plants and controllers. They are briefly described as follows: *PowerInputPowerOutput* represents a port through which the **PlantSystem** connects to the **PowerJunction**. Through it, 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**. Through it, the **PowerJunction** sends the averaged sensor signal to the **ControllerSystem** and receives the calculated control signal from the **ControllerSystem**.

WirelessNetwork Fig. 6 represents the network and its parameters for the NCS. The **WirelessNetwork** model provides modifiable parameters for simu-

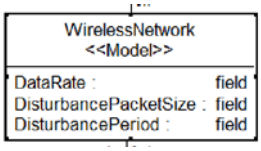


Fig. 6. Wireless Network portion of the Metamodel

lation. *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.

3.2 Language Aspects

Our modeling language has two aspects (GME aspects are similar to modeling views in other tools): **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.
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.

3.3 Structural Semantics

The main objective of our language design is to ensure the "correctness-by-construction" for passive designs of NCS designed using PaNeCS. In order to achieve this objective, we impose constraints on the properties of components of NCS as well as their interconnections. The metamodel notations described above does not capture all the required structural constraints. Using the Object Constraint Language (OCL), we can describe well-formedness rules for defining precise control of static semantics of the language. GME is embedded with an OCL engine which can be used to define constraints that are enforced at design time, giving direct feedback when the user attempts to create faulty connections in the model or violates any of the specified constraints. In Section 4, we will describe an analysis tool that is used to verify that system components satisfy the component-level passivity constraints.

We implemented three classes of constraints: Cardinality Constraints, Connection Constraints and Unique Name Constraints. *Cardinality Constraints* ensure that the required and 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 bidirectional connection between the *Plant* and *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.

An example of an 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 bidirectional connection
between BilinearTransformC to the DigitalController

Equation: let dstCount = ...
 self.attachingConnections("src",Controller_Bilinear)->size in
 dstCount <> 0 implies dstCount = 1
```

3.4 Operational Semantics

NCS modeled in PaNeCS are implemented in MATLAB/Simulink using models generated by an integrated code generator which is discussed in Section 5.

The *Plant* and *DigitalController* entities are implemented as Simulink blocks that model the behavior of each entity based on user-specified parameters. The **PlantSystem** and **ControllerSystem** are modeled as Simulink subsystems. In order to model the behavior of a real network, Simulink is extended with TrueTime as described previously.

Each **PlantSystem** and **ControllerSystem** are connected to a TrueTime Kernel block. The TrueTime Kernel essentially represents each subsystem as a node in the network. It is responsible for I/O and network data acquisition as well as implementing other user-defined tasks, and models the computer or processor on which the subsystem is implemented. The task in each TrueTime kernel connected to a **PlantSystem** is performed periodically based on the specified sampling time of the subsystem. For this version of our language, the **PowerJunction** is implemented as a task in the TrueTime Kernel connected to the **ControllerSystem**. The task that implements the power junction operates based on the occurrence of an event such as the arrival of sensor data or control signal. Each TrueTime kernel has two main scripts: 1) The Initialization script specifies the number of inputs and outputs, the function code name and also indicates the kernel's node id which is used to identify the kernel on the network. 2) The function script essentially implements the user specified task such as sending and receiving of wave variables over a network. The function code for the TrueTime kernel connected to the **ControllerSystem** also performs the additional task of implementing the **PowerJunction**. Hence, the **PowerJunction** sends and receives wave variables from the **ControllerSystem** locally while the **PlantSystem** sends and receives wave variables from the power junction over the simulated wireless network.

The wireless network is implemented using the TrueTime Wireless network block. It simulates the network dynamics, implementing the transfer of data packets over a wireless network from one node to another. It essentially simulates the routing of data received from the TrueTime kernels over the wireless network to their respective destination.

In a typical cycle of operation of the NCS, the wave variables from a **PlantSystem** or multiple **PlantSystem**s are computed and sent to the **PowerJunction** from each TrueTime kernel. The received wave variables are sent to the **ControllerSystem** to compute the control signal which is then sent back to the **PlantSystem**s.

4 Passivity Analysis

4.1 Component Analysis

In order to achieve the desirable properties observed in passive systems, we have to analyze the components of the networked control system and make sure they satisfy passivity constraints.

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 [14, 15]. On invocation (i.e. the modeler presses a button), a C++ model interpreter within GME [13] visits each component, and invokes the analysis function. Any components failing the passivity test are reported to the user.

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, a 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 to solve the LMI for each component.

4.2 System-Level Analysis

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

The realization of the power junction element enforces some simple mathematical constraints which ensure passivity for interconnected components at runtime. These effects are also captured in the simulation of the power junction, so simulation should reveal any destabilizing effects. 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 [6], [9], [17]) of the composed system design.

5 Code Generation

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

We developed 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 [13]. 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.

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 discussed in Section 3 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.

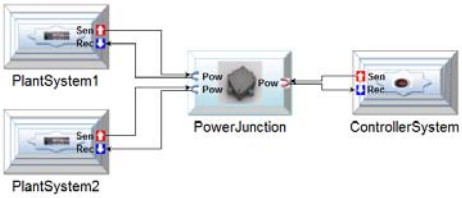
6 Case Study

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.

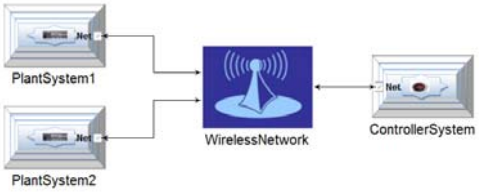
We created a networked control system which involves the control of two discrete plants using a single controller. The controller controls the two discrete plants to track a specified reference signal. The goal of the experiment was to model the network control system and generate a simulation of the behavior of the system. Although we used only two discrete plants for this case study, PaNeCS can model and simulate an arbitrary number of plants.

Fig. 7a and 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 are simple integrators (corresponding to physical models of inertial 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 sampling time, T_s were provided as parameters to the instance model.

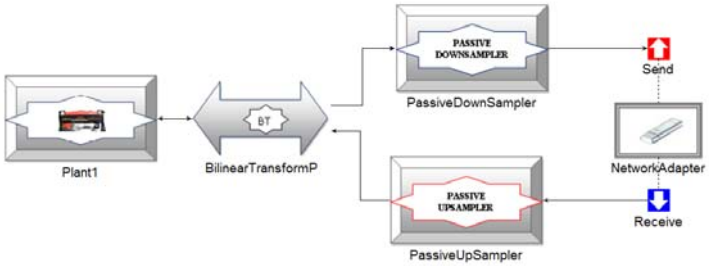
We used a proportional controller as the digital controller to command the plants to track a user-specified reference. The digital controller was also modeled in state space form and the A, B, C and D matrices and also the sampling time, T_s were provided as input parameters to the instance model. The parameters for



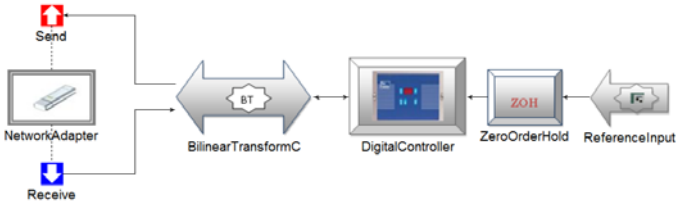
(a) Control Design Aspect



(b) PlatformAspect



(c) Plant Subsystem



(d) Controller Subsystem

Fig. 7. Sample Model of a Networked Control System

the dynamics of the plants and controller are provided in Table 1. The analysis tool checked and verified that the *Plant* and *DigitalController* models satisfied the passivity constraints. Then the code generator was 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. 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. Table 2 shows the parameters for the simulations.

Experiment 1: Nominal Conditions In experiment 1, the system operated without the introduction of disturbance attacks. The three sample periods considered were 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 plots for the results of the NCS having a sample period of 0.1s. Fig. 8 displays the velocity of the plants and the reference velocity provided to the controller. The plants closely tracked the reference velocity. The round trip delay for each plant seemed to have 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	$.1s$

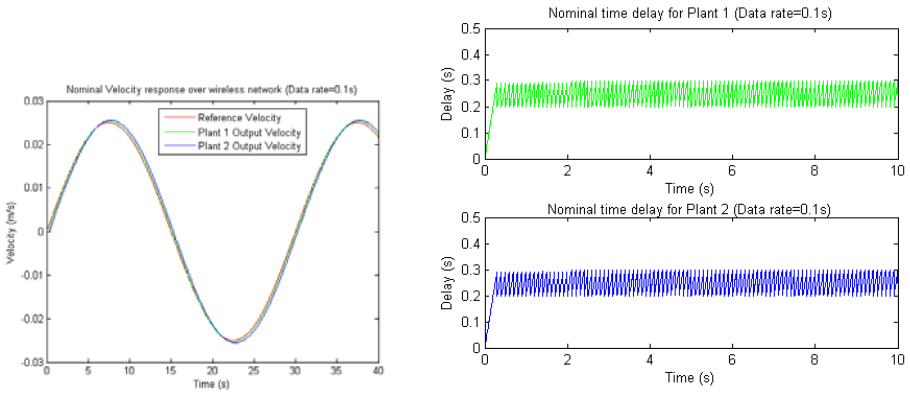


Fig. 8. Nominal velocity response and time delays (Data rate=0.1s)

Experiment 2: Network disturbances In experiment 2, a disturbance attack was introduced in the network. A disturbance node is configured using the *DisturbancePeriod* and DisturbancePacketSize from the **WirelessNetwork** model. Disturbance packets were sent over the network based on the value of a uniformly generated random number. Similar to Experiment 1, three different sample rates were tested, but we only present the results for the 0.1s sample period. 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.

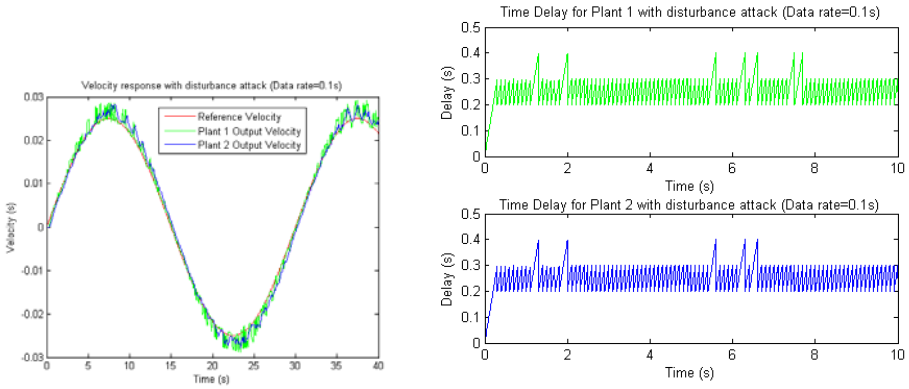


Fig. 9. Velocity response and time delays with disturbance attack (Data rate=0.1s)

Table 2. Simulation Parameters Summary.

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

7 Conclusion and Future Work

Our model-based approach simplifies the process of designing passive networked control systems. We presented PaNeCS, a prototype modeling language for that purpose. We have presented an analysis tool that is used to test system components for passivity. We have also described 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 wireless 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 that a networked control system could be designed using our approach which is robust and insensitive to uncertainties due to a few particular 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. Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: Model-integrated development of embedded software. Proceedings of the IEEE **91**(1) (Jan. 2003)
2. 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
3. AS-2 Embedded Computing Systems Committee: Architecture analysis and design language (aadl). Technical Report AS5506, Society of Automotive Engineers (November 2004)
4. Hudak J. and Feiler P.: Developing aadl models for control systems: A practitioner's guide. Technical Report CMU/SEI-2007-TR-014, CMU SEI (2007)
5. 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)
6. Kottenstette, N., Hall, J., Koutsoukos, X., Antsaklis, P., Sztipanovits, J.: Digital control of multiple discrete passive plants over networks. Intl. Journal of Systems, Control and Communications, Special Issue on Progress in Networked Control Systems (2009)
7. The MathWorks, Inc.: Simulink/Stateflow Tools. http://www.mathworks.com
8. 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/.
9. 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
10. Fettweis, A.: Wave digital filters: theory and practice. Proceedings of the IEEE **74**(2) (1986) 270 – 327
11. Secchi, C., Stramigioli, S., Fantuzzi, C.: Digital passive geometric telemanipulation. In: IEEE Intl. Conference on Robotics and Automation. (2003) 3290 – 3295
12. 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
13. 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)
14. Grant, M., Boyd, S.: Cvx: Matlab software for disciplined convex programming. http://stanford.edu/ boyd/cvx (February 2009)
15. 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
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. 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. RTSS (December 2008) 15–24