

Toward a Science of Cyber–Physical System Integration

This paper focuses on the design phase of the system lifecycle and proposes a passivity-based approach to decouple system stability from cyber timing uncertainties.

By JANOS SZTIPANOVITS, *Fellow IEEE*, XENOFON KOUTSOUKOS, *Senior Member IEEE*, GABOR KARSAI, *Senior Member IEEE*, NICHOLAS KOTTENSTETTE, *Senior Member IEEE*, PANOS ANTSAKLIS, *Fellow IEEE*, VIJAY GUPTA, *Member IEEE*, BILL GOODWINE, *Member IEEE*, JOHN BARAS, *Fellow IEEE*, AND SHIGE WANG, *Senior Member IEEE*

ABSTRACT | System integration is the elephant in the china store of large-scale cyber–physical system (CPS) design. It would be hard to find any other technology that is more undervalued scientifically and at the same time has bigger impact on the presence and future of engineered systems. The unique challenges in CPS integration emerge from the heterogeneity of components and interactions. This heterogeneity drives the need for modeling and analyzing cross-domain interactions among physical and computational/networking domains and demands deep understanding of the effects of heterogeneous abstraction layers in the design flow. To address the challenges of CPS integration, significant progress needs to be made toward a new science and technology foundation that is model based, precise, and predictable. This paper presents a theory of

composition for heterogeneous systems focusing on stability. Specifically, the paper presents a passivity-based design approach that decouples stability from timing uncertainties caused by networking and computation. In addition, the paper describes cross-domain abstractions that provide effective solution for model-based fully automated software synthesis and high-fidelity performance analysis. The design objectives demonstrated using the techniques presented in the paper are group coordination for networked unmanned air vehicles (UAVs) and high-confidence embedded control software design for a quadrotor UAV. Open problems in the area are also discussed, including the extension of the theory of compositional design to guarantee properties beyond stability, such as safety and performance.

KEYWORDS | Control engineering computing; embedded software; system analysis and design

Manuscript received March 30, 2011; revised June 16, 2011; accepted June 23, 2011. Date of publication September 1, 2011; date of current version December 21, 2011. This work was supported by the National Science Foundation under Grant CNS-1035655. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

J. Sztipanovits, X. Koutsoukos, G. Karsai, and N. Kottenstette are with the Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37069 USA (e-mail: janos.sztipanovits@vanderbilt.edu; xenofon.koutsoukos@vanderbilt.edu; gabor.karsai@vanderbilt.edu; nkottens@isis.vanderbilt.edu).

P. Antsaklis and V. Gupta are with the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN 46556 USA (e-mail: antsaklis.1@nd.edu; Vijay.Gupta.21@nd.edu).

B. Goodwine is with the Department of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, IN 37069 USA (e-mail: billgoodwine@nd.edu).

J. Baras is with the Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20740 USA (e-mail: baras@umd.edu).

S. Wang is with General Motors, Warren, MI 48093 USA (e-mail: shige.wang@gm.com).

Digital Object Identifier: 10.1109/JPROC.2011.2161529

I. INTRODUCTION

System integration is currently the largest obstacle to effective cyber–physical system (CPS) design, which is due primarily due to a lack of a solid scientific theoretical foundation for the subject. The absence of a solid understanding of the science of system integration is not due to neglect, but rather due to its difficulty. Most large system builders have given up on any science or engineering discipline for system integration—they simply treat it as a management problem. System integration is almost totally absent from computer science and engineering curricula.

The huge significance of this problem has long been recognized by industry and considered to be a grand challenge. In this paper, we present some preliminary results directed toward a new and comprehensive approach to the subject.

System integration today relies on *ad hoc* methods: After all the components have been designed and manufactured, existing integration methods simply aim at “making it work somehow.” As the complexity of engineered systems continues to increase, our lack of a systematic theory for systems integration creates more and more problems. Finding a solution is difficult because system integration is the phase where essential design concerns—usually separated into physical systems, software, and platform engineering—come together and the hidden, poorly understood interactions and conflicts across design domains suddenly surface. Hence, system integration is particularly challenging in CPS where fundamentally different physical and computational design concerns intersect.

As an example, the automotive industry has been struggling with system integration for many years [1]. Traditional control system development relies on suppliers to provide system components, each implementing a control function using software and hardware. The original equipment manufacturers (OEMs) purchase such system components and integrate them into a vehicle product. Since different suppliers implement their products with different strategies, engineering processes, and tools, OEMs always face the integration challenge, regardless of whether the components are designed and developed for the same vehicle products. The increasing complexity of electrical and control components, and the use of more advanced technologies such as smart sensors and actuators, wireless networks, and multicore processors make the vehicle control integration challenge far worse. OEMs need solid scientific and engineering foundations that enable predictable integration of independently developed system components. Specifically, theories, methods, and tools are required for: 1) design, analysis, and verification of components at various levels of abstraction, including system, software architecture, and controller level, each of which is in its own discipline but subject to the constraints from other levels; 2) composing and analyzing the interactions between vehicle control and physical systems (e.g., engine, transmission, steering, wheel, brake, suspension, etc.) to ensure system-level properties (e.g., stability, safety, performance, cost); and 3) system-level behavior simulation with incomplete or limited information. Know-how in system integration is increasingly the differentiator in competitiveness in automotive as well as other major industrial sectors such as aerospace, health, and defense.

The need for a science of system integration was first expressed over a decade ago in the aerospace community driven by the pressures of rapidly growing complexity of space systems [2], [3]. We believe that in order to address

these problems, significant progress needs to be made toward a new science and technology foundation for CPS integration, one that is model based, precise, and predictable. The unique challenges of a CPS integration science emerge from the heterogeneity of components and interactions inside CPS systems. This heterogeneity drives the need for modeling and analyzing cross-domain interactions among physical and computational/networking domains and demands deep understanding the effects of heterogeneous abstraction layers in the design flow. Transforming system integration from a high-risk management practice into a science-based engineering discipline is a significant challenge.

As CPS-based solutions become ubiquitous, the need for theories, methods, and tools to ensure predictability of system behavior has significantly increased and expanded to most engineering systems. The main contribution of this paper is to present a CPS integration approach to achieving compositionality for a critical system level property—stability—in networked control systems. The results show that 1) passivity-based design of the physical dynamics fully decouples stability from timing uncertainties caused by networking and computation and 2) cross-domain abstractions provide effective solutions for model-based fully automated software synthesis and hi-fidelity performance analysis.

In Section II, the paper describes the impact of heterogeneity and neglected interactions in the loss of compositionality which creates tremendous difficulties in system integration. Section III presents passivity, a powerful system property that allows compositional control design and decouples stability from implementation uncertainties. In Section IV, the inherent robustness that passivity offers against uncertainties is exploited for the design of a networked multiagent system that is stable even in the presence of time-varying network delays and data loss. Section V presents a correct-by-construction method for implementation of networked controllers that includes design of the software components, selection of their interaction model, and design of their execution on a selected implementation platform. Finally, Section VI summarizes the main conclusions from our preliminary work and discusses open research directions.

II. IMPACT OF HETEROGENEITY ON COMPOSITIONALITY

Composition is a technical foundation for all engineering disciplines; it helps to manage complexity, decreases time-to-market, and contains costs. The feasibility of component-based system 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 are not changing as a result of interactions with other components [4]. Lack of compositionality leads

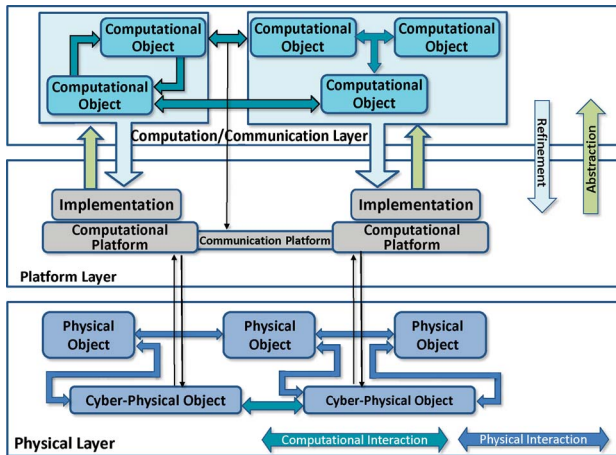


Fig. 1. Design layers in CPS.

to brittleness, that is, systems that do not behave well outside a small operational envelope.

Foundations of component-based design are well understood and successfully applied in many engineering and computer science disciplines, such as digital logic in computer engineering, linear control systems in control engineering, or process algebra in distributed computing—just to name a few. The common feature of all successful compositional design frameworks is homogeneity in terms of the properties to be composed and their semantic domain. It should be noted, however, that though homogeneity is a common feature in successful compositional design frameworks, that property in and of itself does not guarantee compositionality in all cases [5]. Unfortunately, CPSs are inherently heterogeneous in structure, component behavior, and types of interactions among components. CPSs have three fundamentally different design layers: the physical layer and the two “cyber” layers: software and network/platform layers as shown in Fig. 1. For example, the desired dynamics of a car engine and its controller is modeled first with physical layer abstractions. The controller, which is usually implemented as an embedded system, is modeled using software layer abstractions. Finally, the software system is deployed on a computation platform modeled with network/platform layer abstractions. While usually this design process is considered as refinement—adding more details to higher level abstractions—the differences are more profound; the different abstraction layers use fundamentally different semantics and composition concepts.

The physical layer embodies physical components and their interactions which are expressed in continuous (physical) time. Behavior of physical components is determined by physical laws that impose relations on the system variables. Interconnection among physical components occurs through shared variables without a prescribed notion of causality [6]. The mathematics of composition is

based on linear algebra, topology, and analysis. The software layer comprises the software components with behavior expressed in logical time. Interconnections are modeled using various models of computations (MoCs) [7] where software components are connected using an input/output (I/O) model with an implied notion of causality. The mathematics of compositionality is discrete, based on logic, combinatorics, and universal algebra. The network/platform layer comprises the hardware side of CPS architectures and includes the network architecture and computation platform that interact with the physical components through sensors and actuators. While executing the software components on processors and transferring data on communication links, their abstract behavior is “translated” into physical behavior. The behavior of the platforms is modeled by discrete event systems. The mathematics of composition is timed automata, hybrid automata, algebra, and queuing theory.

It is the underlying fundamental differences in the time, concurrency, and composition models that separate the composition principles and verification and validation techniques in the three layers. In general, existing composition frameworks make assumptions that are frequently not satisfied across the layers. This is the result of the phenomenon of “leaky abstractions” [8] when the underlying implementation of a higher level abstraction influences its semantics unintentionally. Problems emerging from cross-layer semantic interactions are ubiquitous. Here are a few.

- 1) Establishing compositionality for properties in physical systems (for example, stability) can be nontrivial even in simple cases. Much of the focus in control theory has been on restricted system architectures, such as feedback loops, model-predictive control architectures, and others, to make the problems solvable. However, control theory on the physical layer tends to neglect effects of implementation of networked controllers on compositionality. For example, effects of uncertain processor availability, probabilistic delays, fixed point arithmetic, coefficient quantization, and signal quantization introduce nonlinearities that may result in limit-cycle oscillation and lack of convergence.
- 2) Dependability, safety, or security properties are frequently satisfied with software that have significant impact on the predictability of timing properties of software applications. For example, multilevel security requirements can be effectively satisfied by using separation kernels. However, separation kernels interfere with timing properties and the resulting increased jitter may lead to unacceptable performance degradation of the system dynamics.
- 3) Uncertainty in wireless communication must be mitigated by the use of complex protocols that

provide improved fault tolerance. These protocols introduce significant time-varying delays in transmissions that can easily destabilize control loops.

The impact of these neglected interactions is the loss of compositionality which creates tremendous difficulty in system integration. The paper presents a theory of composition for heterogeneous systems that can enable scaling up the systems' size and achieve predictable behavior in terms of stability and performance. As opposed to pursuing the development of yet another compositional framework that works only under assumptions (that may not be met in CPS) the paper addresses compositionality in layered systems by introducing decoupling techniques among the design layers. The rationale for decoupling is twofold: 1) it limits system complexity by eliminating intractable cross-cutting interactions among system layers; and 2) it enables making design changes in one layer without influencing essential properties in the others. The cost of decoupling is some level of performance loss relative to a more optimal but brittle design. Effectiveness of the approach in narrower problems (e.g., [9]–[11]) inspired the development of a systematic approach for CPSs described in this paper. Our primary target is to decouple stability—a crucial property on the physical layer—from the effects of implementations on software and network/platform layers.

III. PASSIVITY

Stability is, of course, the primary concern when designing complex systems. In physical processes, stability can be analyzed using energy conservation laws, and the theory of passive dynamical systems provides a strong foundation for a compositional framework for stability. Traditional passive systems theory is a powerful tool for system analysis and control design [12]. Its origins are in electrical circuit theory where networks of passive circuit components were known to be stable in various configurations. These components were often linear, but the general theory of passivity can be applied to general nonlinear systems. Passivity theory has been applied to analysis and design of many systems using a traditional notion of energy. In more general cases, passivity can be applied even when there is not a traditional notion of energy, but rather a generalized energy. This generalized energy can be defined using an energy storage function. When a storage function exists and the energy stored in a system can be bounded above by the energy supplied to the system, the system is passive.

We present a formal definition of passivity [13]. Consider the system H

$$\dot{x} = f(x, u)$$

$$y = h(x, u)$$

where $x \in X \subset \mathbb{R}^n$ is the process state, $u \in U \subset \mathbb{R}^m$ is the control input, and $y \in Y \subset \mathbb{R}^p$ is the output of the system. The system H is said to be passive if there exists a storage function $V(x) \geq 0$ such that $\forall t_1 \geq t_0 \geq 0$, $x(t_0) \in X$ and $u \in U$

$$V(x(t_1)) - V(x(t_0)) \leq \int_{t_0}^{t_1} u^T(\tau)y(\tau) d\tau.$$

Note that if $V(x)$ is differentiable, we can alternatively write

$$\dot{V}(x) \leq u^T(t)y(t) \quad \forall t \geq 0.$$

Intuitively speaking, $V(x)$ refers to the energy content of a system while $u^T(t)y(t)$ refers to the power being fed to the system. Passivity implies that energy is being dissipated from the system. Notice that such an intuition holds exactly for electrical circuits; for more general systems, this is merely a guide.

There are several variations of this definition but essentially all definitions state that the output energy must be bounded so that the system does not produce more energy than was initially stored [14]. Strictly output passive systems and strictly input passive systems with finite gain have a special property in that they are ℓ^2 -stable. In addition if all internal states are *zero-state detectable* then the system is Lyapunov stable [15].

The interest on passivity in CPS follows from the fact that it allows compositional design. Passive systems have a unique property that when connected in either a parallel or negative feedback manner as shown in Fig. 2(a) and (b), respectively, the overall system remains passive. Passivity is preserved since the energy stored in the interconnection is simply the sum of the energies stored in each of the systems, and therefore, a storage function that satisfies the passivity definition for the interconnected system can be easily constructed as the sum of the storage functions of the individual systems. Using these results, large-scale systems can be shown to be stable by verifying passivity for each system component and by following simple interconnection rules. Moreover, a large variety of systems and network components can be made passive. Thus, passivity is an important tool for compositional design of large-scale CPS, particularly when the structure also admits of symmetries.

The theory of passive dynamical systems provides a strong foundation for a compositional framework for stability [14], [16]. Passive systems have infinite gain margin and at least 90° of phase margin, and thus are able to tolerate large loop gains [16]. Passive control theory is very general and broad and applies to a large class of controllers

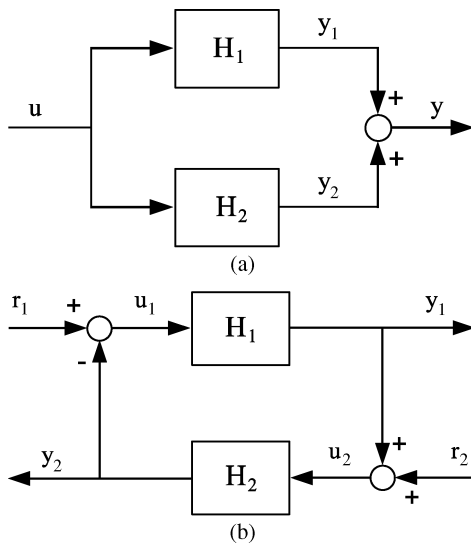


Fig. 2. Interconnections that preserve passivity. (a) Parallel interconnection. (b) Negative feedback interconnection.

for linear, nonlinear, continuous, and discrete-time control systems. The inherent safety of passive systems has been widely exploited by researchers in human interacting machines like smart exercise machines and teleoperated devices (e.g., [17] and [18]). Further, passivity provides significant advantages dealing with network delays, data dropouts, and quantization. The problem of bilateral teleoperation of a single controller/plant pair for both continuous- and discrete-time communication channels is studied in [19]. The approach presented in [20] demonstrates how to interconnect a continuous-time controller to a continuous-time plant while maintaining stability for arbitrary fixed time delays. An approach which allows one to connect discrete-time controllers to continuous port-Hamiltonian plants while preserving passivity even with dropped data and time-varying delays is presented in [21]. A passivity approach for model-based compositional design of an experimental networked multirobot system that ensures the overall system remains stable in spite of implementation uncertainties such as network delays and data dropouts is presented in [22]. Passivity has been used also for group coordination of multiagent systems. A passivity-based design framework for controlling nonlinear systems which have been rendered passive through an internal feedback configuration is presented in [23]. The design procedure is applied to group coordination problems and group agreement (consensus). The framework has been applied to synchronize a group of rigid bodies in [24] and to combine formation control with path following in [25].

As defined above, passivity is only a binary characterization of system behavior based on whether that system dissipates sufficient energy. However, there are systems that dissipate significantly more energy than is required to maintain passivity. Likewise, there are nonpassive systems

that would become passive with a simple loop transformation. This information can be presented in the form of passivity indices [26]. The design advantage of the concept of passivity indices is that even when the system to be controlled is not passive (or even not stable), the passivity indexes can be used to design a feedback system that still renders the interconnection stable [27]. Thus, for instance, if a linear system G_1 is output feedback passive (OFP) with passivity index ρ_1 and input feedforward passive (IFP) with passivity index μ_1 , and another linear system G_2 is nonpassive with OFP(ρ_2) and IFP(μ_2), then their feedback interconnection would still be stable if $\rho_1 + \mu_2 \geq 0$ and $\rho_2 + \mu_1 \geq 0$. Similar results can be derived for nonlinear or switched systems. Moreover, using passivity indices, it can also be checked if the cascade connection of two systems is passive. We have introduced passivity indices that allow consideration of systems that may not be passive, passivity of systems in cascade, design of feedback hybrid passive systems, and methods for designing complex control systems based on symmetries and dissipativity [27]–[29]. These methods can significantly extend the class of CPSs whose design can exploit passivity.

We have investigated the use of passivity for the design of networked control systems, and in particular for decoupling the control design from implementation uncertainties. We exploit the inherent robustness that passivity offers against uncertainties to design a networked multiagent system that is stable even in the presence of time-varying network delays and data loss in Section IV, and we generate control software while ensuring stability is preserved in the presence of jitter in Section V.

IV. DECOUPLING CONTROL DESIGN FROM NETWORK UNCERTAINTIES

Surveillance and convoy tracking applications often require groups of networked agents for redundancy and better coverage. An important goal upon deployment is the establishment of a formation around a target. Although distributed algorithms using only local communication that achieve this goal exist, they typically ignore destabilizing effects resulting from implementation uncertainties, such as network delays and data loss. We address these issues by introducing a discrete-time distributed design framework that uses a compositional, passivity-based approach to ensure l_2^m -stability regardless of overlay network topology, in the presence of network delays and data loss. Furthermore, we show that asymptotic formation establishment and output synchronization can be achieved. Finally, we present simulations of velocity-limited quadrotor unmanned air vehicles (UAVs) to illustrate the performance in the presence of time-varying network delays and varying amounts of data loss.

Performing coordinated tasks in multiagent systems using only local information has been studied extensively over the past decade [30], [31]. Typically, in group

coordination the desired formation emerges from the design of the control law. In [32], the so-called information filter is used for formation stability of LTI systems. For coordination of nonlinear systems, contraction theory with wave variable communication [33], explicit design of Lyapunov vector fields [34], and passivity [23], have been used successfully.

Much of the above work, especially the passivity-based methods, has considered continuous-time systems; however, for digital implementation the design must consider discrete time. In addition, implementation uncertainties such as network delays and data loss must be taken into account. In the following, we present results that demonstrate decoupling between the control design and discrete-time implementation by using a passivity-based framework inspired by work in telemanipulation [19], port-Hamiltonian systems [21], and networked control systems [35].

A. Networked Control System Design

We consider the problem of a multiagent system establishing a formation in \mathbb{R}^2 upon deployment. Assume a global inertial coordinate system and suppose the starting positions of the agents are arbitrary. The goal is to establish an n -gon, where the n agents tend to the coordinates of the vertices asymptotically. Formally, we assign a vertex v_i of the n -gon to agent i , with position $p_i(k)$, $i = 1, 2, \dots, n$. Then, we require $\lim_{k \rightarrow \infty} \|p_i(k) - v_i\|_2 = 0$. We design the networked multiagent system so that the agents converge to a location defined by a reference signal in a synchronized manner in order to achieve some global task. Instead of simply relaying the reference independently to each UAV, we couple their positions in a feedback manner to ensure synchronization of the global task.

In general, the overlay network is bidirectional with asymmetric delays. As a simple example, consider the three node network shown in Fig. 3(a). Each node represents a quadrotor UAV, with each edge modeling the desired communication between UAVs. Realistically, each link in the overlay network is subject to delay imposed by routing, packet handling, and transmission delays. We model such uncertainties using time-varying delays [e.g., $d_{ij}(k)$] as shown in Fig. 3(a). Our objective is to provide a passive-by-construction, discrete-time multiagent network. In the following, we first describe the networked control protocol focusing on the details required for implementation, we then analyze passivity and stability of the networked control system, and finally we present simulation results for distributed deployment of a network of eight quadrotor UAVs.

1) *Networked Control Protocol*: The agent model is shown in Fig. 3(b). Each agent i receives an input reference r_i , which influences the output y_i of the agent through the system mapping H_i , which describes a compensated plant. In this section, we will assume that the system H_i is strictly output passive. In case when the plant is not passive, it may

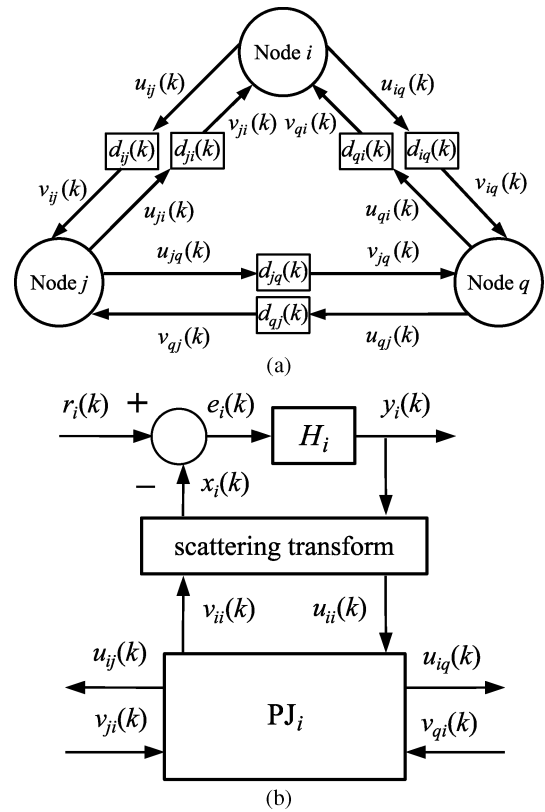


Fig. 3. Network and node structure. (a) A three node network. (b) Node structure.

be possible to render the system strictly output passive by using local compensation (as illustrated for the quadrotor UAV later in this section). As shown in Fig. 3(b), an agent includes two additional components: the *scattering transformation* and the *power junction* PJ_i . The scattering transformation is used to transform the variables x_i and y_i into the wave domain. The node's wave variables u_{ii} and v_{ii} , in turn, are coupled to other nodes through a power junction PJ_i , which allows two or more systems to be connected in a passivity-preserving manner [35].

In distributed control applications, typically the information transmitted across the network has inherent physical meaning. It is well known that transforming these physical variables into the wave domain can preserve passivity and stability for a single bidirectional connection [19] and for star networks [35]. We have extended these approaches to distributed networks with arbitrary overlay topology [36]. For each $i \in V$, the *scattering transformation* produces power waves $u_{ii}(k)$ and $v_{ii}(k)$ defined by

$$u_{ii}(k) = \frac{1}{\sqrt{2b_i}} (b_i y_i(k) + x_i(k))$$

$$v_{ii}(k) = \frac{1}{\sqrt{2b_i}} (b_i y_i(k) - x_i(k))$$

where b_i denotes the characteristic impedance. This definition is similar to the one in [18], with the force and velocity variables replaced with x_i and y_i . Traditionally, the scattering formalism has been applied to power variables (effort and flow) while closing the loop on velocity. Here, the scattering formalism is used abstractly (without the physical interpretation) to close the loop on position. The scattering transformation is treated as a mathematical definition, where the characteristic impedance b_i is defined appropriately for physical consistency (in this case b_i is unitless).

The power junction is a component which allows two or more systems to be connected in the wave domain in a passivity-preserving manner. Formally, let u_i, v_o denote the inputs and v_i, u_o denote the outputs of the power junction. Then, the power junction must satisfy

$$\sum_{i \in \mathcal{S}_{in}} u_i^T(k) u_i(k) - v_i^T(k) v_i(k) \geq \sum_{o \in \mathcal{S}_{out}} u_o^T(k) u_o(k) - v_o^T(k) v_o(k).$$

We can implement the power junction using a set of linear equations that satisfies the above constraints [36]. For each $i \in V$, $j \in N_i$, and $k \in \mathbb{Z}^+$, the outgoing waves are computed as

$$u_{ij}(k) = \frac{1}{\sqrt{n_i}} u_{ii}(k) \quad v_{ii}(k) = \frac{1}{\sqrt{n_i}} \sum_{j \in N_i} v_{ji}(k)$$

where n_i is the number of neighbors of agent i .

In addition, we specify a set of rules for handling network packets. Due to the presence of delays and data loss, some (or all) of the $v_{ji}(k)$ may not be received at time k , in which case we set $v_{ji}(k) \triangleq 0$. Whenever the receiver's buffers are empty, we process null packets. Handling delayed and dropped packets as null packets satisfies the synchronous assumption and preserves passivity [19]. We also constrain the network by preventing retransmission of data for each agent. Based on these assumptions, each channel satisfies the following inequality regardless of time-varying delays and data loss

$$\|(v_{ij})_N\|_2^2 \leq \|(u_{ij})_N\|_2^2, \quad \text{holds } \forall N \in \mathbb{N}.$$

This inequality states that each channel, viewed as the I/O mapping is passive, and therefore does not introduce energy into the system.

2) *Analysis*: Based on the passivity of the interconnections, we can show that a multiagent network designed using the protocol described above is l_2^m -stable for any bidirectional overlay network with asymmetric delays

whenever the I/O mapping of each agent is strictly output passive. Consider a network of n interacting agents implementing the control protocol described above. Then, the following global energy constraint:

$$\sum_{i=1}^n \left(\|(u_{ii})_N\|_2^2 - \|(v_{ii})_N\|_2^2 \right) \geq 0$$

is satisfied for all $N \in \mathbb{N}$, regardless of time-varying delays and data loss. Further, the entire networked system is strictly output passive for arbitrary network topologies. It then follows that the networked control system is l_2^m -stable (proofs of the theoretical results can be found in [36]).

Any system that is strictly output passive is necessarily l_2^m -stable. Therefore, each agent described by H_i is inherently stable. The benefit of the networked control protocol is that it ensures that interactions caused by the networked do not destabilize the networked multiagent system. Passivity holds even in the presence of time-varying delays and data loss.

The coupled multiagent system can establish a desired formation upon deployment by analyzing the behavior at steady state [36]. The system will converge to the ideal steady-state case for constant reference inputs with moderate time-varying delays and data loss because of the disturbance rejecting properties of strictly output passive systems. Suppose that each agent's system mapping admits an invertible steady-state gain matrix G_i (i.e., $y_i = G_i e_i$). The steady-state output of agent i is given by

$$y_i = M_i \left(r_i + \frac{\sqrt{2b_i}}{\sqrt{n_i}} \sum_{j \in N_i} \frac{1}{\sqrt{2b_j n_j}} (r_j + K_j y_j) \right)$$

where

$$M_i = (b_i G_i + I_m)^{-1} G_i \quad \text{and} \quad K_j = (b_j G_j - I_m) G_j^{-1}.$$

Using this transformation, the references for each agent can be computed for the UAVs to surround a target and establish an n -gon.

B. Simulation Results

Our experimental setup involves a network of eight velocity-limited quadrotor UAVs that communicate in a regular overlay network topology. The UAVs move in the plane, and the goal is to form an octagon with each UAV 100 m from a target centered at the origin. The initial points of the UAVs are randomly selected within the region between a 1000-m square and the circle with 100-m radius, both centered at the origin. For simplicity, since

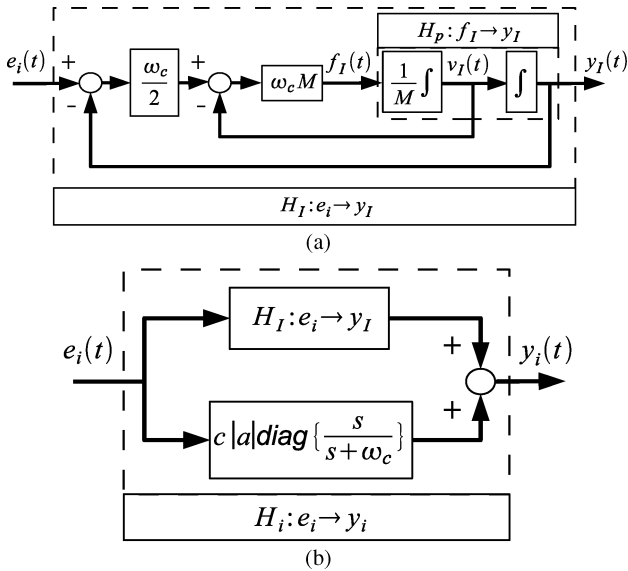


Fig. 4. Simplified quadrotor UAV model. (a) Nonpassive UAV model. (b) Strictly output passive UAV model.

here we focus on coordination between UAVs, we model the quadrotor as a point mass in two dimensions described by

$$\dot{y}_I(t) = v_I(t) \quad M\dot{v}_I(t) = f_I(t).$$

A detailed control design that shows that this is a good approximation when the velocity and the motor thrusts are below the saturation level is presented in [37]. The detailed quadrotor model is used to illustrate the embedded control software design in Section V. For each UAV, we add a local position control system shown in Fig. 4(a). The closed-loop system of the UAV with the local controller is a stable second-order system that is not strictly output passive; however, by adding a high-pass filter in parallel as depicted in Fig. 4(b), the system may be rendered strictly output passive. This model is discretized using a bilinear-like transform with sampling period $T = 0.01$ s, called the inner-product equivalent sample and hold (IPESH) transform, which preserves the passive properties of the system [35]. We also include saturation in the actuators f_{sat} to limit the velocity.

For this example, the overlay network is modeled as an 8-node graph where each node has degree $\eta = 4$. It is assumed that the agents communicate with each other in a synchronous manner. The networked multiagent system is implemented in Simulink [38]. TrueTime is used to simulate the network for communicating between UAVs [39]. The network protocol used is IEEE 802.11b, with a speed of 11 Mb/s. Table 1 shows the values of the parameters used in the example.

Table 1 Parameters for the Network of Quadrotor UAVs

$\omega_c = 6.283$ Hz	$M = 1.826$ kg	$f_{sat} = 40$ N
$c = 2$	$\alpha = -1/2(1 + \sqrt{2})$	$b_1 = 1, G_i = 1$

Scenario 1: No data loss. We consider time-varying delays with a nonuniform constant delay bias in all the communication channels of the network. The delay biases are between 1 and 2 s. We simulate time-varying delays by introducing a disturbance node in the network which floods the network with disturbance packets based on a Bernoulli process with parameter $d = 0.5$. The disturbance node samples a uniformly distributed random variable $X[k] \in [0, 1]$ every 0.05 s. If $X[k] > d$, a disturbance packet is forced on the network. Fig. 5(a) shows the average and maximum errors for the nominal case and the combined delays with no data loss. The results show that the UAVs remain stable and converge to the desired configuration even in the case of time-varying delays.

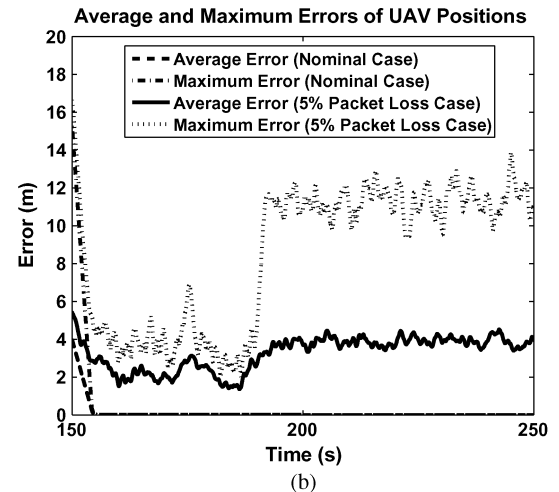
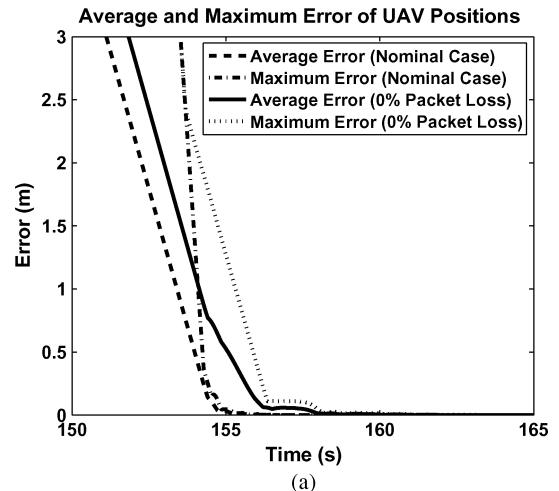


Fig. 5. Results with network delays and data loss. (a) No packet loss. (b) Five percent packet loss.

Scenario 2: 5% data loss. In this scenario, 5% packet loss is introduced with the time-varying delays described above. A probabilistic model based on a Bernoulli process is used to implement the loss of packets independently in each channel. The results, shown in Fig. 5(b), illustrate that with packet loss the system does not achieve zero steady-state error. However, the UAVs still manage to come very close to the desired configuration, demonstrating the resilience of the network. The UAVs end up within an average error of 4 m of the desired configuration.

Our results show that passivity improves decoupling between the controller design and implementation design layers. In particular, we decouple stability of the networked multiagent system from networked implementation uncertainties that result in time-varying delays and data loss. A major concern with passivity-based approaches is that although they can tolerate network uncertainties, they may lead to a conservative design that limits the responsiveness to fast reference signals because of the constraints imposed on the controllers. In our design, we assumed high-frequency signals are filtered and we had to tune the filter and control gains for improving performance. Although the simulations assume ideal models for the delays and the packet loss, our theoretical analysis ensures stability of the networked control system without using such restrictive assumptions. We have also evaluated the approach using an experimental testbed consisting of two robotic manipulators controlled over an 802.11 wireless network in a laboratory environment and we have shown that stability is preserved while performance is degrading because of the unreliable wireless connection [22].

V. DECOUPLING SOFTWARE DESIGN FROM PLATFORM UNCERTAINTIES

Passivity-based design decouples stability, which is a key system-level property, from implementation induced timing uncertainties. In addition to robustness to timing uncertainties, correct-by-construction implementation of networked controllers requires the design of the software components, selection of their possible interactions, and the realization of their execution on a selected implementation platform that leads to further complex challenges. We use here the term “correct-by-construction” to express the goal to eliminate integration-time errors emerging from undesirable cross-layer interactions. In this section, we introduce a framework and a suite of tools for addressing these challenges, we summarize the main steps of the design flow for implementing embedded control software, and we illustrate the more important steps using a quadrotor UAV example.

A. Embedded Systems Modeling Language

1) *Overview:* The Embedded Systems Modeling Language (ESMoL) is a software architecture language for

safety-critical software designs [40], [41]. It shares many concepts with the Architecture Analysis and Design Language (AADL) [42], [43], but differs in that it is a much simpler language, has an intuitive visual notation, and aims to restrict component interactions available to designers of embedded software and systems to functions available on platforms that guarantee inherent safety and fault tolerance properties. Code generators of the ESMoL tool-chain create functional C code from Simulink subsystem blocks (similar to Real-Time Workshop [38]), as well as platform-specific task execution and data communication code for time-triggered (TT) networks. The ESMoL tool-chain differs from the Real-Time Workshop and TargetLink [44] tools in that 1) it makes the architecture modeling explicit; 2) it gives a great degree of control to the designer in allocating the computations and communications to physical resources; 3) it uses a common TT task execution and communication model implemented via a virtual machine (VM); and 4) it creates a schedule for the tasks and messages at design time. The VM relies on static scheduling of the controller network, with the goal of keeping the software for scheduling and execution of the controllers simple enough to analyze formally.

In ESMoL, functional specifications for components appear in the form of Simulink/Stateflow models or as existing C code snippets. The execution of Simulink data flow blocks is restricted to the periodic, discrete-time model which is consistent with the underlying TT platform. The platform definition language supports concepts, relationships, and attributes needed for describing TT networks. The models cover the network topology, as well as various parameters to describe performance parameters like data rates and bus transfer setup times. Parameters for hardware devices also capture timing resolution, overhead for data transfers, and task context switching times.

The component architecture, deployment, and timing/execution models represent different design aspects of the system. Together, the information expressed in the three aspects constitutes a complete model suitable for scheduling analysis, platform-specific simulation, and code generation. The component architecture aspect describes the logical dataflow among component instances. The semantics of this model is based on task-local synchronous function invocations (with shared memory messaging for colocated components) and message transfers between remote tasks using a TT communication scheme. The deployment aspect captures the realization of component instances as periodic tasks running on particular processors. In ESMoL, a task executes on a processing node at a fixed, periodic rate and all components within the task execute synchronously. Data exchanged between tasks take the form of messages in the model. For data transfers, the runtime system supports logical execution time semantics found in TT languages such as Giotto [45].

The timing aspect allows the designer to specify component execution timing constraints. Individual components can be annotated with timing objects that indicate whether they should be executed strictly (i.e., via statically scheduled, TT tasks), or as periodic real-time or as sporadic tasks. Messages are similarly annotated. The annotation objects contain parameters such as the period and worst case execution time that must be provided by the designer. A constraint-based automated scheduling analysis tool calculates the task and message release times. The execution model also indicates which components and messages will be scheduled independently, and which will be grouped into a single task or message object. The temporal order of the message writers and readers is enforced by the static schedule. The locality of a message transfer is specified in the logical architecture and deployment aspects. In the case of processor-local data transfers, transfer time is neglected as reads and writes occur in locally shared memory. After a static schedule has been calculated, the timing objects of the model are back-annotated with the task and message release times. Behavior of the deployed software components depends on the execution times of the functions on the platform, the calculated schedule, and coordination between distributed tasks. The calculated static execution schedule can be used to simulate the control design with the delays introduced by the implementation to assess the impact of the platform effects on performance.

2) *Key Features:* The ESMoL language and tools provide a single multiaspect embedded software design environment so that modeling, analysis, simulation, and code generation artifacts are all clearly related to a single design model. Models appropriate to the different design domains are incorporated in a consistent way using the model-integrated computing (MIC) approach [46]. Models use language-supported relations to associate Simulink control design structures with software and hardware design concepts to define a software implementation for controllers. ESMoL models include objects and parameters to describe the deployment of software components to hardware platforms. Analysis artifacts and simulation models that are generated contain representations of the behavioral effects of the platform on the original design. We include platform-specific simulations to assess the effects of distributed computation on the control design [47].

The integrated analysis, simulation, and deployment capabilities can shorten design cycles. The tool suite includes integrated scheduling analysis tools which converge quickly in most cases so that static schedules can be calculated in rapid design and simulation cycles [48]. We include automatic generation of platform-specific task configuration and data communications code in order to rapidly move from modeling and analysis to testing on actual hardware. Finally, we generate analysis models and code from the intermediate language using simple tem-

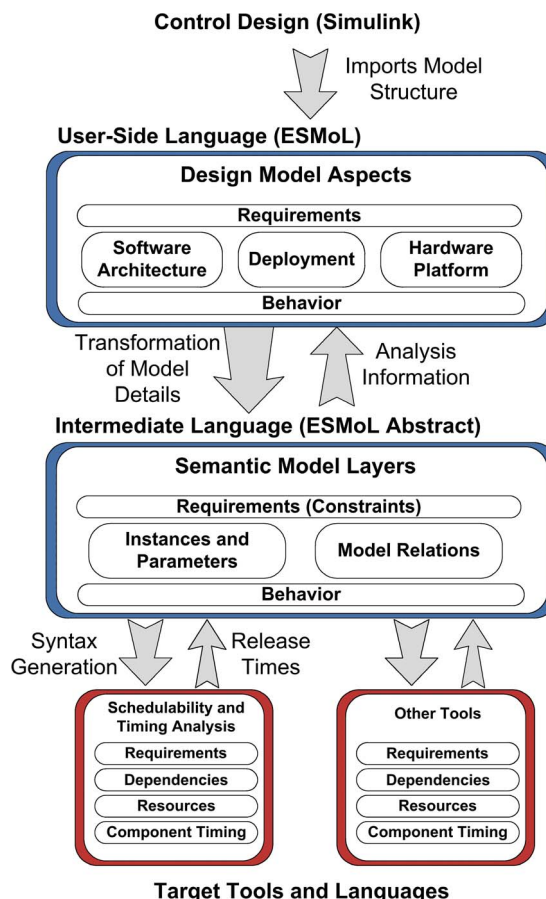


Fig. 6. ESMoL design flow.

plate generation techniques [48]. The incorporation of calculated schedule analysis results back into the ESMoL model helps to maintain consistency as models pass between design phases.

B. Embedded Control Software Design Flow

Fig. 6 depicts the basic ESMoL design flow. The software designer imports an existing Simulink control design into the generic modeling environment (GME) [49], configured to edit ESMoL models, and encapsulates the subsystem blocks imported from Simulink into software components that will implement the controllers. The designer also creates models for the hardware architecture of the TT platform. Finally, the designer instantiates components to create multiaspect models that specify logical dependencies, hardware deployment, and timing models. A completed model is transformed into a model in the ESMoL_Abstract language, resolving all implied relationships and structural model inferences. Model interpreters are used to integrate schedulability and timing analysis tools. Finally, the tool chain allows the generation of platform-specific simulations of the system implementation as well as deployable code. Potential targets include

the TrueTime toolkit for platform-specific simulations in Simulink [39], [47] and the portable TT FRODO virtual machine [50]. In the following, we describe in more detail the main steps of the design flow.

The controller implementation starts with the specification of the controller dynamics at the physical layer (see Fig. 1). This specification usually takes the form of an I/O-oriented physical model (for example, in Simulink/Stateflow). The model can be defined by a 5-tuple $M_P = \langle A, D, \phi, \omega, < \rangle$. A is a set of activities implemented by software primitives. An activity $a_i \in A$ represents a finite set of computation steps that performs a mapping between its input and output data. D is a set of data exchanges among activities. An elementary data exchange $d_i \in D$ occurs among a subset of activities by communicating a finite set of encoded symbols. The functions $\phi : A \rightarrow T$ and $\omega : D \rightarrow T$ assign to the activities and data exchanges a (usually) periodic timing requirement expressed as activity and message activation rate. Finally, $<$ is an irreflexive transitive binary relation on A representing data dependencies among the activities.

An essential choice for the implementation design is the model of computation (MoC) that defines the semantics of interactions among software components. A key issue in networked embedded controllers is meeting the time constraints defined by ϕ and ω . Satisfaction of these time constraints must be independent from the underlying network/platform model, therefore an MoC that decouples timing requirements on the software layer from the actual platform-dependent execution times and communications times is the preferred solution. Accordingly, we have selected the TT architecture [51] where activities and data exchanges occur according to a precalculated periodic schedule. If the performance of the computation platforms and communication channels is sufficient for satisfying the scheduling requirements then the overall system operation remains correct.

With these considerations the implementation progresses in the following steps.

Step 1: Import the controller design M_P from the physical layer in the software implementation environment.

The source controller model from Simulink is imported to the ESMoL modeling environment [40]. The dataflow semantics of the original Simulink model is preserved and is fully represented within the ESMoL model. Software components in ESMoL are defined by creating references to Simulink subsystem blocks and to input and output messages. Instances of these software components correspond to individual runtime tasks. Each task has logical execution time semantics [52], which means that all input messages are available before a task consuming them is released, and output messages of the task are sent at precisely defined points in time, after the task has finished. Message types and their data elements are also defined.

Step 2: Design of a network/platform architecture model $H = \langle P, L \rangle$ specified as a set of computation nodes,

where L is a set of communication channels and a channel $l_j \in L$ connects a subset of computation nodes $l_j \in P \times P$.

The ESMoL modeling environment includes a network/platform architecture modeling language [41] using the level of abstraction sufficient for representing software component and message deployment decisions.

Step 3: Design of a software component architecture $S = \langle C, M \rangle$ where the set of software components C is a partition over A , and a message $m_k \in M$; $m_k \in C \times C$ is a subset of $d_i \in D$ data exchanges among activities that belong to different components.

Once components and message types are defined, the components and messages are translated into platform-independent functional code. The internal dataflow representation of each component is converted into C-code blocks which are executed on top of a thin virtual machine that implements the TT execution semantics.

Step 4: Design of mappings $\delta : C \rightarrow P$ and $\gamma : M \rightarrow L$ specifying deployments of software components and messages to communication channels.

The ESMoL modeling environment provides a deployment model view for component and message allocation. Multiple instance of a particular type of component may be deployed. Messages are mapped to communication channels via which they will be communicated.

Step 5: Synthesis of schedulers for software components and messages controlling the access to processor and communication resources such that timing requirements ϕ, ω in the physical model M_P are satisfied.

The ESMoL design environment includes a scheduler that synthesizes the task and a communication schedule for the distributed platform. Generation of the schedule requires to use cross-layer abstractions that include modeling concepts from all three design layers. The abstracted specifications are limited to information about platform connectivity, assignment of tasks to processors, routing of messages through buses, and timing specifications. Platform-specific performance is represented by worst case execution time (WCET) and worst case communication time (WCCT). The specifications are mapped into a finite-domain integer constraint problem that models dependencies between components (tasks) and messages, exclusive use of processors and buses, and timing constraints (i.e., maximum acceptable latency between tasks). If the problem is solvable, then a solution will satisfy the specified timing requirements and will contain start times for each of the tasks and messages [40].

Step 6: Verification of performance requirements for the overall physical dynamics including the effects of implementation.

Our current approach for verifying performance requirements is to synthesize a high-fidelity simulation model that refines the physical dynamics model M_P with new components modeling essential effects of the controller implementation. We use the TrueTime extension of Simulink [39] for this purpose. The TrueTime models are

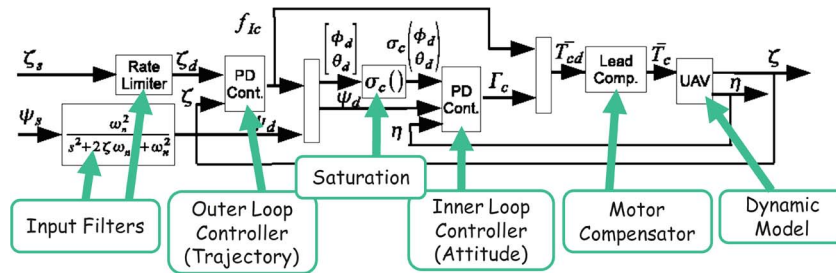


Fig. 7. Control architecture for the quadrotor control problem.

generated from the software layer and network/platform layer models using automated model transformations [47].

The TT execution model is used only for running the closely coupled physical systems components (such as the flight control system of a quadrotor UAV including processor nodes and communication buses). Loosely coupled system components that communicate via wireless networks (such as a group of UAVs) interact without a global communication schedule. As it was described in the previous section, the passivity-based design can be used to ensure stability of the vehicle formation, but the overall performance needs to be verified using simulation.

C. Quadrotor UAV Example

We demonstrate our method by generating the control software for a quadrotor helicopter. Quadrotor helicopters are lifted and propelled by four rotors. This architecture provides agility and high degree of maneuverability. Unlike traditional helicopters, they do not require a tail rotor to control yaw and can use four smaller fixed-pitch rotors. By having smaller rotors, these vehicles can achieve higher velocities before blade flapping effects begin to destabilize it and limit performance.

Fig. 7 depicts the control architecture. We design two proportional-derivative (PD) controllers. The inner-most loop controller is a “fast” PD attitude controller in which attitude is described by Euler angles. Next, we close the loop with a second PD inertial (trajectory) controller. Because there is a significant lag between rotor thrust commands and the resulting change in thrust due to the acceleration of the air columns above their respective rotors [53], we add an additional lead compensator to minimize this effect. The rotors can only apply a fixed range of thrust due to motor driver voltage limits which requires limiting the range of attitude commands using a saturation function. We also limit the maximum velocities by adding a position rate change limiter to the desired inertial position setpoint. The rate change limiter includes an additional second-order prefilter applied to minimize overshoot and a similar filter is applied to the yaw setpoint. The detailed control design that includes justification for the passive PD controllers and proof of stability can be found in [37]. Further, the work in [37] shows that the

proposed control design ensures that the controlled quadrotor UAV can be approximated by a point mass model (as the model used in Section IV) as long as the velocity and the motor thrusts are below the saturation levels.

Our approach described above aims to address cross-domain interactions. We require stability from the software-implemented closed-loop system over the full range of possible inputs. Performance should exhibit reasonable tracking of a reference trajectory within the physical limits of the vehicle. Simulation-based analysis of the physical system yields execution rates for the software control components. Passive design provides a guarantee of stable operation around a nominal sampling rate, as the system will tolerate a small number of fixed sampling delays or of missed data samples.

The controller design provides task periods and static analysis or code profiling provides execution time specifications for each component instance. Data transfer rates and overhead parameters are captured in the platform model. The scheduling process guarantees that the implementation meets the timing requirements required by the control design (provided the task execution times are honored). Scheduling problem specifications include data dependencies, resource allocation, and end-to-end latency requirements.

The quadrotor helicopter that we used is the AscTec Hummingbird AutoPilot from Ascending Technologies [54]. The quadrotor’s hardware architecture consists of low level processor board that interfaces with the inertial measurement unit (IMU) sensors and the motors and a high level processor board that is used for the control software. Both are based on the Philips LPC2146 processor, which implements the ARM7TDMI-S specification, and include two universal asynchronous receiver/transmitter (UARTs), serial peripheral interface (SPI), synchronous series port (SSP), and other peripherals.

Fig. 8 shows the software components. Sensor_Convert is the component that converts longitude and latitude received by the global positioning system (GPS) to local coordinates. Outer_Loop is the component for inertial position control and Inner_Loop for attitude control. Reference_Handler is used as a reference trajectory.

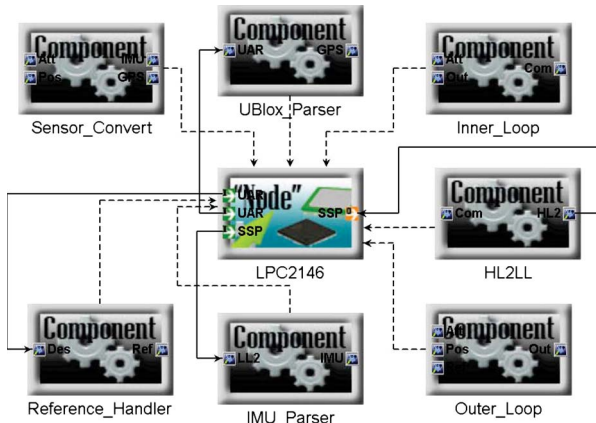


Fig. 8. The deployment model of the controller software components.

UBlox_Parser, IMU_Parser, and HL2LL enable the communication with the low level board by coding and decoding sensor and command data. The model assigns each software component to its own real-time task. In Fig. 8, the dashed connection from a component to a node reference represents an assignment of that component to run as a task on the node. The port connections represent the hardware channel through which that particular message will travel. Local data dependencies are not specified here, as they are represented in the logical software architecture. I/O channel objects (ports) on the node block can also be connected to message objects on a component. These connections represent the flow of data from the physical environment through sensors (input channels) or the flow of data back to the environment through actuators (output channels).

Each software component must have timing information to fully specify its execution behavior. This includes execution period and WCET as its parameters. Each external data transfer is similarly annotated for TT communication. For the processor-local data messages, transfer time is neglected, as reads and writes occur in local, shared memory. The quadrotor helicopter platform provides a fundamental sampling rate of 1 kHz. The *ExecPeriod* attribute for all components is set as shown in Table 2. The

Table 2 Analyzed WCET and Stack Usage for Each Component

Component Name	WCET (μ s)	Stack Usage (B)	Sampling Rate (Hz)
Outer_Loop	299	176	100
Inner_Loop	163	96	100
Sensor_Convert	70	52	100
Reference_Handler	0.45	4	100
UBlox_Parser	68.5	44	10
IMU_Parser	19	40	333
HL2LL	29.75	36	333

fundamental rate required for the controller is 100 Hz. Sensor and actuator data rates drive the other components. For example, since the time between two valid GPS samples is 100 ms, the *ExecPeriod* for *Blox_Parser* is also 100 ms, because it processes the GPS data. The worst case latency from sensors to actuators must be smaller than 10 ms. Local message transfers may be specified as TT, but in practice they take place in shared memory and are not scheduled. The worst case latency from all sensors to any affected actuators must be smaller than 1 ms.

The memory system consists of 256-KB on-chip flash memory (ROM) and 32-KB SRAM (RAM). The corresponding binary code is about 110 KB, so it fits in the system’s ROM space. All the data variables for the communication are preallocated, and from the table we can see the maximum stack usage of a component is 176 B. None of the components preempt each other during normal execution, so storage is not a concern.

VI. CONCLUSION AND RESEARCH DIRECTIONS

In this paper, we argued for the systematic development of a science for integrating CPSs that is based on investigating compositionality in heterogeneous systems. It has been widely recognized that heterogeneity of system components and interactions is a source of emergent behavior in large-scale systems. However, it is significantly less understood that heterogeneity of abstraction layers used in the design flows also leads to loss of predictability of system behavior as a result of cross-layer, semantic interactions that are neither modeled nor anticipated. Similarly to the importance of establishing composition and compositionality for selected properties inside abstractions layers (usually called horizontal composition), compositionality needs to be established across abstraction layers as well (vertical composition). This paper focuses on vertical composition using the approach of decoupling crosscutting interactions. Full decoupling across abstraction layers means that established properties in individual layers will be preserved independently from changes and uncertainties in other design layers.

We have demonstrated this research approach in the context of networked control system design. We have shown that stability of complex dynamic systems can be decoupled from timing uncertainties induced by network and platform effects. We have also shown that the TT model of computation has a similar role in decoupling software layer behavior from platform-specific effects. Starting from the work presented in the paper, research is needed in many directions, some of which are listed below.

- 1) Passivity-based approaches need to be extended from continuous dynamics to discrete event system models as well. The extension would result in increased robustness in a wider scale of systems,

such as large-scale distributed real-time systems applications, manufacturing systems, and traffic management systems.

- 2) Stability is only one of the essential system properties. Preserving compositionality in other properties, such as safety, and performance requirements are also an important and essential design goal.
- 3) Tool support for model-based integration of CPS requires significant future research effort. Heterogeneous system context requires tool reuse and rapid construction of domain-specific tool

chains, therefore work in this direction is much required.

- 4) Experimental research to validate the scientific results of the theoretical work as well as the engineering tools and processes. ■

Acknowledgment

The authors would like to thank J. Porter, G. Hemingway, H. LeBlanc, E. Eyisi, Z. Zhang, M. McCourt, P. Wu, and H. Yu that have contributed to various aspects of this work.

REFERENCES

- [1] A. L. Sangiovanni-Vincentelli, "Quo vadis SLD: Reasoning about trends and challenges of system-level design," *Proc. IEEE*, vol. 95, no. 3, pp. 467–506, Mar. 2007.
- [2] K. L. Bellman, "Knowledge-based integration infrastructure and the emerging integration science," in *Proc. 15th Int. Congr. Cybern.*, Namur, Belgium, Aug. 1998.
- [3] K. L. Bellman and C. Landauer, "Towards an integration science: The influence of Richard Bellman on our research," *J. Math. Anal. Appl.*, vol. 249, no. 1, pp. 3–31, 2000.
- [4] G. Gossler and J. Sifakis, "Composition for component-based modeling," *Sci. Comput. Programm.*, vol. 55, no. 1–3, pp. 161–183, 2005.
- [5] B. Goodwine and P. J. Antsaklis, "Multiagent coordination exploiting system symmetries," in *Proc. Amer. Control Conf.*, 2010, pp. 830–835.
- [6] J. C. Willems, "The behavioral approach to open and interconnected systems," *IEEE Control Syst. Mag.*, vol. 27, no. 6, pp. 46–99, Dec. 2007.
- [7] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neundorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity—The ptolemy approach," *Proc. IEEE*, vol. 91, no. 1, pp. 127–144, Jan. 2003.
- [8] J. Spolsky, *Joel on Software*. Berkeley, CA: APress, 2004.
- [9] P. Mosterman, J. Sztipanovits, and S. Engell, "Computer-automated multi-paradigm modeling in control systems technology," *IEEE Trans. Control Syst. Technol.*, vol. 12, no. 2, pp. 223–234, 2004.
- [10] J. Skaf and S. Boyd, "Analysis and synthesis of state-feedback controllers with timing jitter," *IEEE Trans. Autom. Control*, vol. 54, no. 3, pp. 652–657, Mar. 2007.
- [11] A. Bhave and B. Krogh, "Performance bounds on state-feedback controllers with network delay," in *Proc. 47th IEEE Conf. Decision Control*, Dec. 9–11, 2008, pp. 4608–4613.
- [12] G. Zames, "On the input-output stability of time-varying nonlinear feedback systems, part one: Conditions derived using concepts of loop gain, concavity, and positivity," *IEEE Trans. Autom. Control*, vol. AC-11, no. 2, pp. 228–238, Apr. 1966.
- [13] J. C. Willems, "Dissipative dynamical systems part i: General theory," *Arch. Rational Mech. Anal.*, vol. 45, no. 5, pp. 321–351, 1972.
- [14] A. J. van der Schaft, *L₂-Gain and Passivity in Nonlinear Control*. New York: Springer-Verlag, 1999.
- [15] D. Hill and P. Moylan, "Stability results for nonlinear feedback systems," *Automatica*, vol. 13, no. 4, pp. 377–382, 1977.
- [16] W. M. Haddad and V. S. Chellaboina, *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Princeton, NJ: Princeton Univ. Press, 2008.
- [17] P. Y. Li and R. Horowitz, "Control of smart machines—Part 1: Problem formulation and nonadaptive control," *IEEE/ASME Trans. Mechatron.*, vol. 2, no. 4, pp. 237–247, Dec. 1997.
- [18] G. Niemeyer and J.-J. E. Slotine, "Telemanipulation with time delays," *Int. J. Robot. Res.*, vol. 23, no. 9, pp. 873–890, 2004.
- [19] N. Chopra, P. Beresteky, and M. W. Spong, "Bilateral teleoperation over unreliable communication networks," *IEEE Trans. Control Syst. Technol.*, vol. 16, no. 2, pp. 304–313, Mar. 2008.
- [20] S. Hirche, T. Matiakis, and M. Buss, "A distributed controller approach for delay-independent stability of networked control systems," *Automatica*, vol. 45, no. 8, pp. 1828–1836, 2009.
- [21] S. Stramigioli, C. Secchi, A. J. van der Schaft, and C. Fantuzzi, "Sampled data systems passivity and discrete port-Hamiltonian systems," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 574–587, Aug. 2005.
- [22] X. Koutsoukos, N. Kottenstette, J. Hall, E. Eyisi, H. LeBlanc, J. Porter, and J. Sztipanovits, "A passivity approach for model-based compositional design of networked control systems," *ACM Trans. Embedded Comput. Syst., Special Issue on the Synthesis of Cyber Physical Systems*, 2011.
- [23] M. Arcak, "Passivity as a design tool for group coordination," *IEEE Trans. Autom. Control*, vol. 52, no. 8, pp. 1380–1390, Aug. 2007.
- [24] H. Bai, M. Arcak, and J. T. Wen, "Rigid body attitude coordination without inertial frame information," *Automatica*, vol. 44, no. 12, pp. 3170–3175, 2008.
- [25] I.-A. F. Ihle, M. Arcak, and T. I. Fossen, "Passivity-based designs for synchronized path-following," *Automatica*, vol. 43, no. 9, pp. 1508–1518, 2007.
- [26] J. Bao and P. L. Lee, *Process Control*. New York: Springer-Verlag, 2007.
- [27] H. Yu and P. J. Antsaklis, "A passivity measure of systems in cascade based on passivity indices," in *Proc. 49th IEEE Conf. Decision Control*, 2010, pp. 4608–4613.
- [28] M. J. McCourt and P. J. Antsaklis, "Control design for switched systems using passivity indices," in *Proc. Amer. Control Conf.*, 2010, pp. 2499–2504.
- [29] M. J. McCourt and P. J. Antsaklis, "Stability of networked passive switched systems," in *Proc. 49th IEEE Conf. Decision Control*, 2008, pp. 1263–1268.
- [30] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proc. IEEE*, vol. 95, no. 1, pp. 215–233, Jan. 2007.
- [31] R. Olfati-Saber, "Flocking for multi-agent dynamic systems: Algorithms and theory," *IEEE Trans. Autom. Control*, vol. 51, no. 3, pp. 401–420, Mar. 2006.
- [32] J. A. Fax and R. M. Murray, "Information flow and cooperative control of vehicle formations," *IEEE Trans. Autom. Control*, vol. 49, no. 9, pp. 1465–1476, Sep. 2004.
- [33] W. Wang and J.-J. Slotine, "Contraction analysis of time-delayed communications and group cooperation," *IEEE Trans. Autom. Control*, vol. 51, no. 4, pp. 712–717, Apr. 2006.
- [34] D. A. Lawrence, E. W. Frew, and W. J. Pisano, "Lyapunov vector fields for autonomous unmanned aircraft flight control," *J. Guid. Control Dyn.*, vol. 31, no. 5, pp. 1220–1229, 2008.
- [35] N. Kottenstette, J. Hall, X. Koutsoukos, P. Antsaklis, and J. Sztipanovits, "Digital control of multiple discrete passive plants over networks," *Int. J. Syst. Commun. Control*, vol. 3, no. 2, pp. 194–228, 2011.
- [36] H. LeBlanc, E. Eyisi, N. Kottenstette, X. Koutsoukos, and J. Sztipanovits, "A passivity-based approach to deployment in multi-agent networks," in *Proc. 7th Int. Conf. Inf. Control Autom. Robot.*, 2010, vol. 1, pp. 53–62.
- [37] N. Kottenstette and J. Porter, "Digital passive attitude and altitude control schemes for quadrotor aircraft," in *Proc. 7th IEEE Int. Conf. Control Autom.*, ChristChurch, New Zealand, 2009, pp. 1761–1768.
- [38] The MathWorks, Inc., *Simulink/Stateflow Tools*. [Online]. Available: <http://www.mathworks.com>
- [39] M. Ohlin, D. Henriksson, and A. Cervin, "TrueTime 1.5 Reference Manual," Dept. Autom. Control, Lund Univ., Lund, Sweden, Jan. 2007. [Online]. Available: <http://www.control.lth.se/truetime/>
- [40] J. Porter, G. Hemingway, H. Nine, C. vanBuskirk, N. Kottenstette, G. Karsai, and J. Sztipanovits, "The ESMoL language and tools for high-confidence distributed control systems design—Part 1: Language, framework, and analysis," ISIS, Vanderbilt Univ., Nashville, TN, Tech. Rep. ISIS-10-109, 2010. [Online]. Available: <http://www.isis.vanderbilt.edu/sites/default/files/ESMoLTOpt.pdf>
- [41] J. Porter, G. Karsai, P. Volgyesi, H. Nine, P. Humke, G. Hemingway, R. Thibodeaux, and J. Sztipanovits, "Towards model-based integration of tools and techniques for embedded control system design, verification, and implementation," *Workshops and*

- Symposia at MoDELS 2008 (ACES-MB)*. Berlin, Germany: Springer-Verlag, 2009, ser. Lecture Notes in Computer Science.
- [42] AS-2 Embedded Computing Systems Committee, "Architecture analysis and design language (AADL)," Society of Automotive Engineers, Tech. Rep. AS5506, Nov. 2004.
- [43] J. Hudak and P. Feiler, "Developing AADL models for control systems: A practitioner's guide, CMU SEI, Tech. Rep. CMU/SEI-2007-TR-014, 2007.
- [44] dSpace, Inc., *dSpace TargetLink: Automatic Production Code Generator*. [Online]. Available: <http://www.dspaceinc.com>
- [45] T. A. Henzinger, C. M. Kirsch, M. A. Sanvido, and W. Pree, "From control models to real-time code using Giotto," *IEEE Control Syst. Mag.*, vol. 23, no. 1, pp. 50–64, Feb. 2003.
- [46] J. Sztipanovits, G. Karsai, S. Neema, and T. Bapty, "The model-integrated computing tool suite," in *Model-Based Engineering of Embedded Real-Time Systems*, vol. 6100, H. Giese, G. Karsai, E. Lee, B. Rumpe, and B. Schätz, Eds. Berlin, Germany: Springer-Verlag, 2011, pp. 369–376, ser. Lecture Notes in Computer Science.
- [47] G. Hemingway, J. Porter, N. Kottenstette, H. Nine, C. vanBuskirk, G. Karsai, and J. Sztipanovits, "Automated synthesis of time-triggered architecture-based TrueTime models for platform effects simulation and analysis," in *Proc. 21st IEEE Int. Symp. Rapid Syst. Prototyping*, Jun. 2010, DOI: 10.1109/RSP.2010.5656335.
- [48] J. Porter, G. Karsai, and J. Sztipanovits, "Towards a time-triggered schedule calculation tool to support model-based embedded software design," in *Proc. 7th ACM Int. Conf. Embedded Softw.*, 2009, pp. 167–176.
- [49] A. Ledeczki, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. T. IV, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The generic modeling environment," in *Proc. Workshop Intell. Signal Process.*, May 2001.
- [50] R. Thibodeaux, "The specification and implementation of a model of computation," M.S. thesis, Dept. Electr. Eng. Comput. Sci., Vanderbilt Univ., Nashville, TN, May 2008.
- [51] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. IEEE*, vol. 91, no. 1, pp. 112–126, Jan. 2003.
- [52] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: A time-triggered language for embedded programming," *Proc. IEEE*, vol. 91, no. 1, pp. 84–99, Jan. 2003.
- [53] G. M. Hoffmann, H. Huang, S. L. Waslander, and C. J. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *Collection of Technical Papers—AIAA Guid. Navig. Control Conf.*, 2007, vol. 2, pp. 1670–1689.
- [54] *AscTec Hummingbird With AutoPilot User's Manual*. [Online]. Available: <http://www.ascotec.de>

ABOUT THE AUTHORS

Janos Sztipanovits (Fellow, IEEE) graduated (*summa cum laude*) from the Technical University of Budapest, Budapest, Hungary, in 1970 and received the Ph.D. degree from the Hungarian Academy of Sciences, Budapest, Hungary, in 1980.

He is currently the E. Bronson Ingram Distinguished Professor of Engineering at Vanderbilt University, Nashville, TN. He is founding director of the Institute for Software Integrated Systems (ISIS). His current research interests include the foundation and applications of model-integrated computing for the design of cyber-physical systems.

Xenofon Koutsoukos (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 2000.

He is an Associate Professor in the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, and a Senior Research Scientist in the Institute for Software Integrated Systems (ISIS). His research interests are in the area of cyber-physical systems with emphasis on formal model-based methods, diagnosis and health management, and sensor networks.

Prof. Koutsoukos was the recipient of the National Science Foundation (NSF) CAREER Award in 2004.

Gabor Karsai (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from Vanderbilt University, Nashville, TN, in 1988.

He is Professor of Electrical Engineering and Computer Science at Vanderbilt University, Nashville, TN, and a Senior Research Scientist in the Institute for Software Integrated Systems. He conducts research in model-integrated computing.

Nicholas Kottenstette (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, IN, in 2007.

He is a Research Scientist at the Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN. His research develops constructive resilient networked control principles for cyber-physical systems.

Panos Antsaklis (Fellow, IEEE) received the Ph.D. degree from Brown University, Providence, RI.

He is the Brosey Professor of Electrical Engineering and Concurrent Professor of Computer Science and Engineering and of Applied and Computational Mathematics and Statistics at the University of Notre Dame, Notre Dame, IN.

Prof. Antsaklis is the Editor-in-Chief of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL.

Vijay Gupta (Member, IEEE) received the Ph.D. degree in electrical engineering from the California Institute of Technology, Pasadena.

He is an Assistant Professor in the Department of Electrical Engineering, University of Notre Dame, Notre Dame, IN. His research and teaching interests lie in the general area of intersection of control, communication and processing.

Prof. Gupta won the National Science Foundation (NSF) CAREER award in 2009.

Bill Goodwine (Member, IEEE) received the M.S. and Ph.D. degrees in applied mechanics from the California Institute of Technology, Pasadena, in 1993 and 1998, respectively. He also has a Juris Doctor degree from Harvard Law School, Cambridge, MA.

He is an Associate Professor in the Department of Aerospace and Mechanical Engineering, University of Notre Dame, Notre Dame, IN.

Prof. Goodwine has been a National Science Foundation (NSF) CAREER Award recipient.

John Baras (Fellow, IEEE) received the Ph.D. degree in applied mathematics from Harvard University, Cambridge, MA.

He is a Professor with the Department of Electrical and Computer Engineering, University of Maryland, College Park, where he holds the Lockheed Martin Chair in Systems Engineering, and he is also a member of the Applied Mathematics, Statistics and Scientific Computation Program Faculty, and Affiliate Professor in the Fischell Department of Bioengineering. His research interests include control, communication and computing systems, model-based systems engineering, and cyber-physical systems.

Shige Wang (Senior Member, IEEE) received the Ph.D. degree in computer science and engineering from the University of Michigan, Ann Arbor, in 2004.

He is a Senior Research Scientist at General Motors R&D, Warren, MI. His current research interests include system modeling, analysis, and codesign of embedded real-time control systems.