Issues in the convergence of control with communication and computing: Proliferation, architecture, design, services, and middleware

Scott Graham, Girish Baliga, and P.R. Kumar

***Abstract*— We anticipate that a possible next phase in the information technology revolution could be the convergence of control, i.e., sensing and actuation, with communication and computing. We address the broad set of issues that we believe to be important to the design, implementation, and proliferation of such systems. In particular, we expound on the topics of the architecture of such systems, methodologies for design, distributed time, services, and middleware. We describe our efforts in the Convergence Lab at the University of Illinois with respect to each of these topics.**

I. Introduction

Over the past fifty years, we have witnessed advances in the fields of communication, computation, and control. As these fields have converged, they have combined strengths and provided new capabilities. Convergence of computation with control has led to digital control systems. Convergence of communication with computation has resulted in the Internet.

This paper is addressed at what we envision could be the next phase of the information technology revolution, the convergence of control with communication and computation, which opens great possibilities for interaction with the physical world. We envision widespread usage, such as efficient traffic scheduling, energy efficient heating and cooling, and home automation. Such systems require sensors to sense the environment and actuators to act upon it, with controllers directing the actuators, and all components linked through communication capability.

Two technological trends making feasible large systems of this sort are the growth in embedded computers and wireless networking. About 98% of all microprocessors sold are embedded [1]. They are present in virtually all automobiles, and many appliances, in addition to the well-established presence in computer, communication, and entertainment equipment. Wi-Fi (IEEE 802.11x) has experienced double-digit growth since 2000 [2]. Lower cost wireless connectivity is possible with Bluetooth, available at a $6 per chipset cost to manufacturers [3]. Extrapolating these trends in wireless communication, we can envision a time, not far off, in which wireless connectivity is a commodity.

With each embedded device functioning as a sensor or an actuator, wirelessly connected with others, the future could well see orchestras of sensors and actuators playing over the ether in vast interconnected control systems. Sensor networks, which can be formed with nodes such as the Berkeley MOTES [4], are becoming a reality. The logical next step is to incorporate actuators and influence the physical world, leading to control. The particular focus in this paper is on what is required to make this vision a reality, and for such systems to proliferate.

The contributions of the paper are as follows. We address the importance of the architecture in the proliferation of such systems. We outline a methodology for designing such systems with the understanding that systems are always evolving. We address the issue of building in reliability from the ground up and specifically illustrate it with respect to our testbed. Then we discuss services that should be provided to such systems in order to make design and implementation routine. One critical service is a timing service, as time is important in control systems that interact with the physical world. We advocate such services being provided by a middleware, which facilitates system upgrades and configuration management by encapsulating low level details such as IP addresses. We also address the issue of migration for self-optimization and dynamic load balancing. This not only improves performance but it also renders system design easier.

A. General-purpose control systems

Our goal is to enable the proliferation of distributed control systems into many applications. We are therefore interested in "general-purpose control systems." Katevenis [5] refers to general-purpose computer systems as "systems not biased towards the execution of a particular algorithm, and specifically, systems that execute a mix of word processing, database applications, mail and communications, compilations, CAD, control, and numerical applications."

We similarly define general-purpose control systems as systems not biased towards a particular control applica-

This material is based upon work partially supported by DARPA/AFOSR under Contract No. F49620-02-1-0325, AFOSR under Contract No. F49620-02-1-0217, USARO under Contract Nos. DAAD19-00-1-0466 and DAAD19-01010-465, and NSF under Contract Nos. NSF ANI 02-21357 and CCR-0325716, and DARPA under Contact Nos. N00014-0-1-1-0576 and F33615-0-1-C-1905. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the above agencies, the United States Air Force, Department of Defense, or the U.S. Government.

Major Graham is studying under U.S. Air Force sponsorship through the AFIT/CI program in the Department of ECE, University of Illinois, 1308 West Main Street, Urbana, IL 61801, USA. srgraham@uiuc.edu

Girish Baliga, CSL and Department of CS, University of Illinois, 1308 West Main Street, Urbana, IL 61801, USA. gibaliga@uiuc.edu

P.R. Kumar, CSL and Department of ECE, University of Illinois, 1308 West Main St., Urbana, IL 61801, USA. prkumar@uiuc.edu

tion, but designed to support many applications such as home automation and control, manufacturing, transportation, robotics, etc. Our goal is to distill the common elements and components necessary for control into an easily customizable and reconfigurable framework for general use. We believe that doing so, and allowing applications to be rapidly produced, is important for the proliferation of such systems.

B. The Convergence Laboratory at the Univ. of Illinois

This lab was developed to identify and investigate what the important issues are in the convergence of control with communication and computation. It features a concrete example of a general-purpose control system. The laboratory has an indoor track upon which remotely controlled cars are driven. (Movies available at [6]).

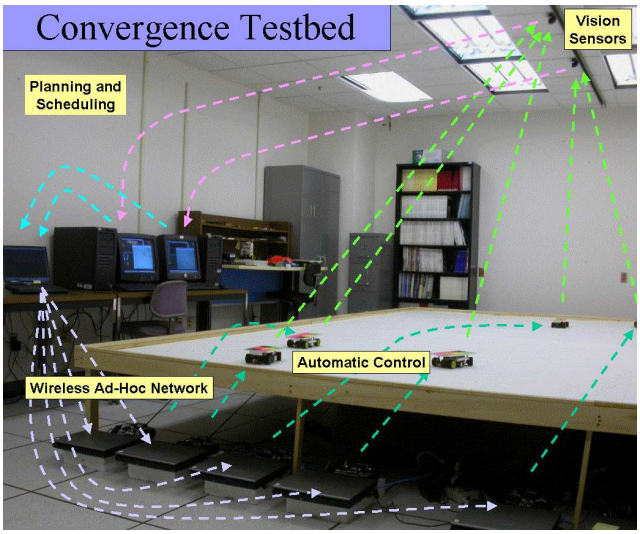

Fig. 1. The Convergence Laboratory at the University of Illinois

1) Sensors: Two ceiling mounted cameras continuously capture images which are converted into an HSV image representation more immune to brightness variations over the track. Pixels are grouped as colors, which are in turn grouped as potential cars. The tops of the cars are color coded, and cars are identified by their unique color patterns. The position and orientation information is determined for each car and distributed via an ad-hoc wireless network.

2) Actuators: The actuators are simple versions of automobiles with two degrees of freedom, steering and velocity. Remote control is through dedicated radio-control frequencies for each car that can be regarded as virtual "wires."

3) Controllers: Each car has a dedicated controller operating at 10 Hz. Incoming sensor data is fused with past control inputs in a state estimation module in order to provide reliable periodic state estimation to the controller.

The control law is Model Predictive Control (MPC) wherein the controller has a model of how the car will move when given particular commands. Searching through a large space of control sequences, the controller performs an optimization at each iteration, choosing the control which minimizes the distance between the desired trajectory and the predicted trajectory over a short time horizon.

4) Control hierarchy: Residing in the layer above the real-time model predictive controller for each car, the Planner provides a desired trajectory aperiodically to the real-time controller. The Planner monitors the positions of other cars possibly re-planning the desired trajectory. As the Planner is collocated with the real-time controller, it provides local autonomous control for fail-safe purposes.

C. Importance of architecture

We believe that the architecture of general-purpose control systems will be key to their proliferation. In general, the usefulness of any overarching design depends on how well it adapts to the particular needs of an individual context, while at the same time capturing the essence of that which is common across most of the usages [8], [9]. As the particular useful variations of a product may not be known to the designers beforehand, we believe that successful proliferation will depend on providing the right abstractions and architecture for use by designers.

As an instantiation, one may consider the OSI model of network protocols. This model separates the various functions of communication into layers of abstractions, with services provided by lower layers to higher level applications or protocols. Design at each layer can be oblivious to lower layers. With interfaces between layers which are well-defined and understood by both sides of the interface, it becomes possible to make changes to an intermediate layer in the protocol stack without affecting the other layers, allowing for incremental evolution. It also gives the design longevity. Indeed, we contend that the reason for the proliferation of the Internet is primarily architectural, and only secondarily due to the algorithms involved.

As another example, one may argue that the success of serial computation is due to the von-Neumann bridge [10]. It allows hardware designers (e.g., Intel) and software designers (e.g., Microsoft) to work separately. As long as they conform to an abstraction of the other party, the products will interoperate. In fact, it is the possible lack of a von-Neumann bridge equivalent that has led to the failure of earlier efforts in parallel computation.

As yet another example, the separation of source coding and channel coding, established by Shannon [11] as an architecture within which ϵ-optimal throughput can be attained, is at the heart of the digital communication revolution. Source coding is done through software (e.g., MPEG or JPEG), while channel coding is done in the network interface cards.

This leads to what we believe is a fundamental question: What is the appropriate architecture and what are the right abstractions for the convergence of control with communication and computing? Our goal is to design an application independent and context independent architecture, as far as

that is possible. If one replaces car specific code by aircraft specific code, the same architecture should support air traffic control. Or if we replace aircraft specific code with code for a thermostat, it should support building temperature control.

D. Importance of middleware

For distributed applications in general, and distributed control systems in particular, there are many low level services which are common across applications. Important to our endeavor is to develop middleware, which resides between the operating system and applications, as a layer enabling the application to be oblivious to details which need not concern it, while at the same time being able to fully utilize the services provided by the underlying hardware and software.

The system designer, the minimization of whose time we believe is of paramount importance to the proliferation of general-purpose control systems, should be supported by the middleware so that she can work at the appropriate domain specific level of detail, concentrating on the big picture of the system. The middleware, by virtue of its position in the application-to-hardware hierarchy, is capable of addressing many of these details. We are therefore interested in the specific requirements of middleware that arise in the context of general-purpose control systems, as well as developing such a middleware.

II. Reliable design in an ever changing environment

Mass production typically separates design from production. Design is completed before components are produced. While this model is very efficient for producing vast quantities of a product, it is not well-suited for general-purpose control for several reasons. First, new technological capabilities will become available during the development of the system, and users and designers must be able to incorporate them. Second, requirements themselves inevitably change as a system is built. Indeed we believe that general-purpose control system designs will always be in a continuous state of evolution, following a spiral model of development [12].

A. Reliability through dependence reduction

To ensure reliability, we must create robust components at every level, with some degree of autonomy to operate, at least in a fail-safe mode for a short period of time, in the presence of failure of other system components. We call this Local Temporal Autonomy. For example, if a controller waits for a new data sample before computing new commands, then it is dependent on the sensor (and the communication channel) for its continued operation. We call this *execution dependence*. More generally, execution dependence exists when the failure of one module causes another module to fail. In this example, the controller is also dependent on the sensor and communication channel for timing. This represents a *timing dependence* of the controller on the sensor and communication channel.

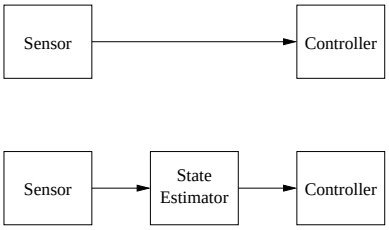

Fig. 2. A mediating interface between incongruent models

To remove the execution and timing dependencies of the controller on the inherently unreliable communication channel, we interpose a module, such as a state estimator [13], as in Figure 2, which accepts aperiodic sensor data as input, and provides continuous or periodic outputs to the controller even in the complete absence of sensor information for some period of time. Architecturally, the above state estimator represents a buffer between dissimilar components. That is, the controller prefers a periodic signal, but the communication channel can at best provide a perturbed periodic signal. With the state estimator serving as a mediating interface between incongruent models, the sensor, communication channel, and controller are free to operate and evolve as needed, both at run-time and over the system lifecycle.

B. Safety through analytic redundancy

Similarly, we must design for safety beginning at the lowest level of control, the actuator, and continuing up the control hierarchy. As an example, in the testbed, the cars are designed to stop after not receiving commands for some pre-determined amount of time. At higher levels, the problem may be posed as one of using a complex controller whenever possible, reverting to a simpler, but more robust controller when the complex controller fails in some way. The ability to do this enables the use of complex, but unverified, components without depending on them. This is similar to the Simplex architecture [14]; see Figure 3.

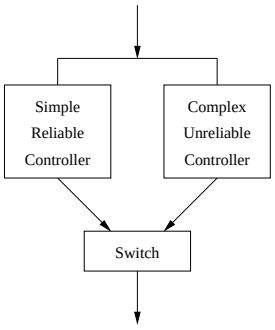

Fig. 3. The Simplex switching architecture

C. Reliability in testbed design

We have built reliability into the testbed in many places.

1) Controller reliability: Robustness against controller failures is enabled by three design elements. First, a windowed horizon of future controls (naturally provided by Model Predictive Control.) Second, sending this window of controls, in addition to the current control, to the actuator for future use, even though most of these will be overwritten when the next sequence arrives. Third, the use of a separate computer process, perhaps on a separate machine, such that the actuator will not fail in conjunction with a controller failure.

2) Reliability through rapid restart: With the design noted above, after a period of open loop control, the cars are timed out for safety. This has, however, provided a new capability in the testbed, namely rapid restart. By checkpointing desired trajectories and start times, a failed controller can be restarted quickly, and resume operation. Such a restart capability has been implemented [15]; Figure 4 shows that restarts have no noticeable effect.

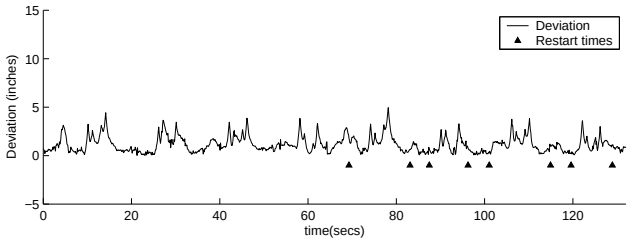

Fig. 4. Deviation from desired trajectory under restart

III. The importance of time

A. The Control Time Protocol

A fundamental problem in distributed systems is that no two clocks agree. Timing discrepancies can cause instability in an otherwise stable control system. The proper knowledge of time is thus critical to control systems which must operate in real-time. There are many challenges to distributed time including the need to avoid introducing dependencies which degrade reliability, several of which are discussed in a companion paper [16], which presents the Control Time Protocol (CTP). In addition to other services, it provides reliability and robustness against inherent timing problems in communication systems, such as latency and jitter, and can enhance the performance of a system.

B. Migration

It is desirable to adapt to the evolving computing and communication loads over the network. For example, it may become feasible to process an image right at the camera, outputting perhaps the location of colors instead of raw vision images which could stress the communication network. This is an optimization function that the system should automatically do at runtime, relieving the designer of the need to even consider such issues. Thus we are interested in automatic migration which adapts to the computational or communication loading. It contributes to the longevity of the code since it need not be updated as components are upgraded.

A timing service, can also provide run-time measurements of processing and propagation times. Using these online estimates, the middleware can optimize its operation by migrating computational loads to underutilized system resources as we describe in Section V.

IV. Layers of control

When multiple vehicles operate in a shared space, we become concerned with collisions. This is a system constraint, such that the control layer immediately above the real-time controller can be regarded as the constraint or "coherence" layer.

In the testbed, this layer is distributed. Individual cars have planners to monitor for collisions and take fail-safe action if necessary. In addition, we have a global scheduler [7] capable of scheduling all cars in collision free trajectories, with provable freedom from deadlock. This is done according to a goal, which is the top layer in our control hierarchy.

The separation of real-time and constraint layers also represents a time-scale separation [17]. Global scheduling may proceed on a slower time-scale than local control. Moreover, system goals may also change aperiodically, typically occurring more slowly than the run-time evolution of the system.

Clearly this control hierarchy can be nested. From one fleet of autonomously controlled cars, we can extrapolate to multiple fleets, each with a fleet goal translated into individual car goals. Thus the hierarchy can be composed into a larger hierarchy, with some higher level entity distributing fleet goals to each fleet.

Thus control hierarchy can be characterized by scope, timescales, and authority. The top levels have the most broad authority, while lower levels are responsible for local decisions which may need to be made in short timescales in response to rapidly changing events. Important decisions which need not be handled quickly can be percolated up to the top of the hierarchy and decided there.

V. Middleware

For the proliferation of general-purpose control systems, it is important to provide appropriate services which free the system designer to design at a higher level. It should be made feasible to create a system from composable components. Composability requires well-defined interfaces, which may not be common across all platforms. A useful solution to this problem is to insert a layer in the application stack which creates a common interface for applications above it, and with respect to the lower level details of the infrastructure beneath it. This is known as middleware. (CORBA [18] is a well-known example of such a middleware.)

Middleware is a widely used class of application frameworks that hide a lot of low level detail, and provide

simpler as well as higher level usage models. Applications using middleware are typically developed as a set of components. Since the middleware supports a lot of common functionality, the components mainly implement application logic such as control laws. Further, well defined interfaces and usage models enable these components to be reused in different applications. For example, a component implementing a path planning algorithm developed for our testbed could be reused in an air traffic control system or the air-conditioning system of a building. Basic tasks such as time-translation can be incorporated in a common application framework that can be reused in multiple applications. This approach saves much development effort, better incorporates accumulated experience, and improves software quality.

A. Requirements

Services required for general-purpose control include the following:

1) Communication: A lot of detail attending communication between application components can be encapsulated in middleware. Addressing needs to be by content aware addressing, wherein the physical location of processes in the system is no longer tied to a particular host machine. Tasks such as deciding where a particular component is to be executed, and details such as the IP address and protocol needed to communicate with it, need not be known. For example, a car controller might wish to obtain feedback from a sensor covering location (27,49). If IP addresses are embedded into this request, then whenever a node is changed to one with a different IP address, all the code has to be checked and references to IP addresses changed (a la the Y2K problem). Computational resources must be able to be changed or updated with little impact on the system, simplifying configuration management.

2) Infrastructure: Middleware can provide templates for rapid insertion of simple functionality into an existing infrastructure. Thus, an entire system can quickly be connected in an early version, allowing the system connectivity issues to be handled before adding complex functionality.

Startup, configuration, monitoring and maintenance of application components, requires an infrastructure "layer" with primitives to automate these tasks. For example, at system start up, a controller may have to wait until a sensor is available, or an actuator is up, before it can begin to operate. Starting these components in the right order should be the function of middleware. Changing modes or parameters of a controller, such as the operating frequency or level of detail in response to availability of resources is another example of functionality that the middleware must provide.

3) Evolution: As noted earlier, systems are continually upgrading or evolving. For example, if a controller is receiving noisy inputs from a sensor, it must be possible to insert a filtering component between these two, without having to restart the system. Tasks such as redirecting the sensor updates to the filter and filter output to the controller, without causing system instability, must be available as primitives.

Using the Control Time Protocol, described in [16], and by time stamping all communication throughout the system, the middleware can keep an accurate, updated estimate of the time on all distributed clocks as well as profile the system, providing per packet delay estimates, or average delay and jitter for messages etc.

The act of moving a computational process from one location to another is *migration*. Correct migration requires properly capturing the state of the system, installation of appropriate code on the new machine, starting the process up on the new machine, and properly shutting down the old process. Doing so safely in a control environment requires adherence to a variety of time considerations. First, proper interpretation of time must be made by the new machine. Second, the delays experienced in the new control loop must be satisfactory. Indeed, one reason for migration is to reduce the overall system loop delay to improve the performance of the system.

B. Etherware

Traditional middleware such as CORBA and its variants have their focus on other domains, such as business and transaction applications. Consequently, many of the design considerations and optimizations are not suitable for networked and wireless control.

The CORBA footprint is quite large, even for very small applications. Moreover, the entire system is based upon TCP, which provides reliable delivery among other services. For control, some data streams are more sensitive to delay than loss, rendering TCP inappropriate for such data streams.

To specifically address the above requirements and optimize for networked control, we have developed *Etherware*, a middleware for networked control applications. Details about *Etherware* and a comparison with standard middleware are provided in [19].

VI. Concluding remarks

While individual distributed control systems exist, most applications are custom designed and manufactured for particular purposes. We seek to move beyond one-of-a-kind systems toward general-purpose control. Applications will emerge, rather than be well-defined from the beginning, requiring flexibility in the design process.

As we are interested in the proliferation of general-purpose control systems, there are a few key requirements common to all systems in this domain.

First, proliferation necessitates rapid and simplified design. Many applications of ubiquitous control could be simple, and perhaps unworthy of excessive design. We must respect the designer's time and remove unnecessary burden. For any widespread proliferation, success will depend on making implementation simple enough so that all that is

required is a domain expertise, but not expertise in computing and communication per se, or in systems integration.

Second, reliability is primary as a performance criterion. Systems must be reasonably robust to perturbations in distributed components, including changing and updating portions of the system over time.

Third, control systems will continuously evolve as new requirements arise as a result of new technologies or the discovery of new applications. Thus, the design of the system will never be complete.

While it may be too early to be definitive about the history of the last fifty years, it can be argued that the last half of the twentieth century was the age of the building of strong disciplines. In computation, the ENIAC [20] and the stored program are about fifty years old. Information theory of Shannon [11] is also of similar vintage, as is also Wiener's "yellow peril" [21]. The state-space control work of Kalman [13] dates to around 1960. Signal processing, at least the work of Cooley and Tukey [22] dates to around 1965.

However, with advances in embedded systems, wireless communication, etc., we are building systems which combine sensing, wireless communication and computing, and actuation will necessarily follow. This will lead to the convergence of control with communication and computation, where all the above individual disciplines become inseparable. For example, the problem of data fusion in sensor networks is not just an inference problem, or just a computation problem, or just a communication problem. It is a synthesis [23]. When actuation is also involved, as in control, there is a further convergence of theories. Such a systems theory could well be the agenda for the next two decades.

Much of the technology required for deployment of general-purpose control systems is readily available. We have outlined here the organizing principles, methodologies for design, design infrastructure, infrastructure of services, and software, which will enable rapid and robust development. These are the issues that are the focus of our efforts at exploration and implementation in the Convergence Laboratory.

REFERENCES

[1] J. Stankovic, "Vest: A toolset for constructing and analyzing component based operating systems for embedded and real-time systems," Tech. Rep. TRCS-2000-19, University of Virginia, July 2000.

[2] K. Carter, A. Lahjouji, and N. McNeil, "Unlicensed and unshackled: A joint OSP-OET white paper on unlicensed devices and their regulatory issues," May 2003. http://hraunfoss.fcc.gov/edocs_public/attachmatch/DOC-234741A1.doc.

[3] V. Lipset, "In-car bluetooth to grow beyond telephony, study says," May 23 2003. http://www.thinkmobile.com/Everything/News/00/67/32/.

[4] CrossBow Technology, Inc., "Wireless sensor networks." http://www.xbow.com/Products/Wireless_Sensor_Networks.htm.

[5] M. Katevenis, "Reduced instruction set computer architectures for VLSI." ACM Doctoral Dissertation Award, MIT Press, 1984.

[6] "The convergence laboratory testbed." University of Illinois. http://black1.csl.uiuc.edu/∼prkumar/testbed/.

[7] A. Giridhar and P. R. Kumar, "Scheduling traffic on a network of roads." Submitted to IEEE Transactions on Vehicular Technology. http://black1.csl.uiuc.edu/∼prkumar/ps_files/trafficpaper.ps, April 17 2003.

[8] C. Alexander, *The Timeless Way of Building*. Oxford University Press, 1779.

[9] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA: Addison-Wesley, 1995.

[10] L. G. Valiant, "A bridging model for parallel computation," *Communications of the ACM*, vol. 33, August 1990.

[11] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–423, 1948.

[12] B. Boehm, "A spiral model of software development and enhancement." ACM SIGSOFT Software Engineering Notes, August 1986.

[13] R. E. Kalman, "On the general theory of control systems," in *Proceedings of the 1st IFAC Congress*, vol. 1, (Moscow), pp. 481–492, Butterworth's, London, 1960.

[14] L. Sha, R. Rajkumar, and M. Gagliardi, "The simplex architecture: An approach to building evolving industrial computing systems," in *Proceedings of the International Conference on Reliability and Quality in Design*, (Seattle, WA, Anaheim, CA), pp. 122–126, ISSAT Press, March 16-18 1994.

[15] S. Kowshik, H. Ding, G. Baliga, S. Graham, and L. Sha, "Managing restarts in networked control systems: A case study." http://black1.csl.uiuc.edu/∼prkumar/testbed/papers/. University of Illinois at Urbana-Champaign fkowshik,huiding,gibaliga,srgraham,lrsg@uiuc.edu, December 2003.

[16] S. Graham and P. R. Kumar, "Time in general-purpose control systems: The control time protocol and an experimental evaluation." Submitted to IEEE Conference on Decision and Control, December 2004.

[17] S. B. Gershwin, *Manufacturing Systems Engineering*. Englewood Cliffs: Prentice-Hall, 1994.

[18] CORBA/IIOP Specifications, "OMG," December 2002.

[19] G. Baliga, S. Graham, L. Sha, and P. R. Kumar, "Etherware: Domainware for wireless control networks." To appear in IEEE Computing, December 2004. http://dsonline.computer.org/0306/f/bal.htm.

[20] J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*. San Mateo, CA: Morgan Kaufamnn Publishers, Inc., 1990.

[21] N. Wiener, "The extrapolation, interpolation and smoothing of stationary time series with engineering applications," 1949.

[22] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, p. 297, April 1965.

[23] A. Giridhar and P. R. Kumar, "Data fusion over sensor networks: Computing and communicating functions of measurements." Submitted to Journal on Selected Areas in Communications, December 2003.

[24] V. Kawadia and P. R. Kumar, "A cautionary perspective on cross layer design," technical report, University of Illinois at Urbana-Champaign, June 28 2003. http://black1.csl.uiuc.edu/∼prkumar/ps_files/Cross_Layer.ps.