Proceedings of the
44th IEEE Conference on Decision and Control, and
the European Control Conference 2005
Seville, Spain, December 12-15, 2005

Middleware and Abstractions in the Convergence of Control with Communication and Computation[⋆]

Girish Baliga[*], Scott Graham[**] and P. R. Kumar[†]

Abstract— **Control technology is entering what may arguably be called its third computational generation, featuring powerful computational facilities, general purpose wire-line or wireless networks, powerful computational services, and large numbers of possibly distributed sensor and actuators: For the development and proliferation of such control systems it appears that it is important to address three challenges. What are the appropriate abstractions and what is the corresponding architecture of such systems? What are the middleware services that are needed for ease of application development and deployment? What are the appropriate theories to exploit the capabilities of these systems? This tutorial is aimed at addressing these three issues. The Etherware middleware is described, an abstraction of virtual collaboration is identified, and it is detailed how Etherware supports the abstraction. The principle of local temporal autonomy is described to address the need for reliability and robustness. These issues are also illustrated on the experimental and development testbed in the Convergence Laboratory at the University of Illinois. Open theoretical problems are identified, along with speculation on what a future theory may entail.**

I. Introduction

The purpose of this tutorial is to identify, formulate, and address issues that are important to the next generation of control systems. Arguably, these control systems can be regarded as "third generation" systems. The first generation of control systems can be regarded as analog control. This was accompanied by theories such as that of Bode [1] or Evans that was important for the technology. The second generation of control systems can be regarded as comprising of digital control. This too was accompanied by theory, such as that due to Kalman [2], [3], [4], which was well matched to the technological needs. In particular, this second generation saw the algorithm itself as the solution of the problem, along with modest computation, as is well exemplified by the discrete-time Kalman filter. Control theory has, arguably, mainly focused on the possibilities afforded by digital control for about the past forty years. However, due to the rapid technological changes of the past decade, there is the possibility of deploying distributed control systems consisting of sensors and actuators connected by shared wired or wireless networks, and involving powerful computational nodes as well as software services. This tutorial addresses the issues related to how to facilitate the proliferation of such next generation control systems.

In the next sections we address the issues of middleware, abstractions, and architecture, and the need for theory, to support development and deployment of such systems. We begin with a description of the laboratory testbed.

II. The Convergence Laboratory

The Convergence Laboratory at the University of Illinois features a number of radio controlled cars on a plywood platform. Ceiling mounted video cameras monitor the platform. The images are processed to determine the locations and orientation of the cars. There are several computers connected by a wire-line Ethernet or IEEE 802.11 wireless ad hoc network. Each car is individually controlled by a laptop over a dedicated radio frequency. Figure 1 shows the physical experimental platform.

Through the methods to be described in the sequel, several operational application scenarios are feasible. In one "city traffic" mode, safety (avoiding collisions) as well as liveness (guaranteeing that each car reaches its destination without deadlock) are provided via provably correct algorithms. In a "collision avoidance" mode,

[⋆]The research reported here has been supported by NSF under Contract Nos. NSF ANI 02-21357 and CCR-0325716, USARO under Contract Nos. DAAD19-00-1-0466 and DAAD19-01010-465, DARPA/AFOSR under Contract No. F49620-02-1-0325, DARPA under Contact Nos. N00014-0-1-1-0576 and F33615-0-1-C-1905, and AFOSR under Contract No. F49620-02-1-0217.

[*]CSL and Dept. of CS, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. email: gibaliga@uiuc.edu.

[**]Maj. Scott Graham, Air Force Institute of Technology AFIT/ENG, 2950 Hobson Way, WPAFB, OH 45433. email: Scott.Graham@afit.edu.

[†]CSL and Dept. of ECE, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA. email: prkumar@uiuc.edu.

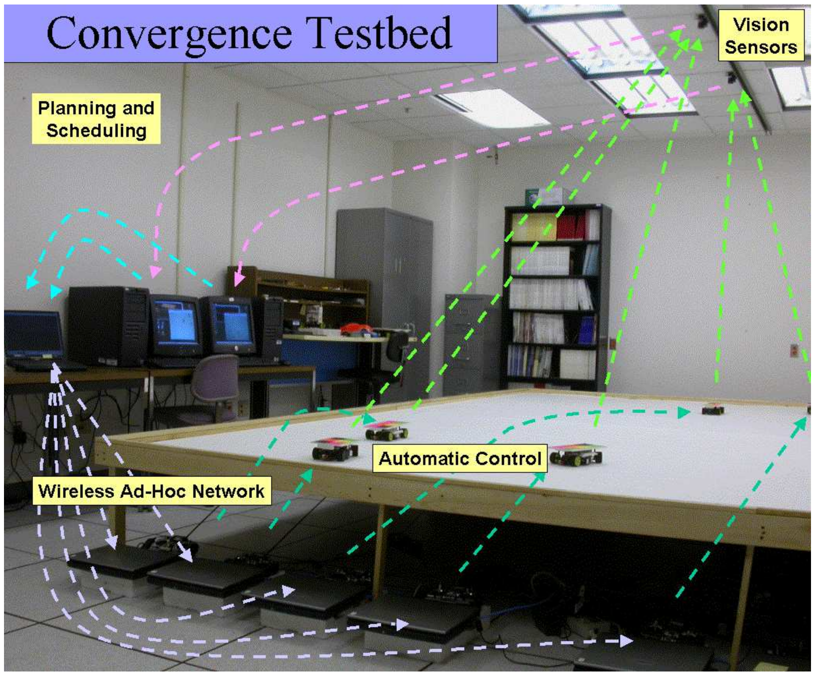

Fig. 1. Convergence Laboratory experimental testbed

cars automatically resort to safety maneuvers to avoid head-on collisions. In another "pursuit-evasion" scenario, a human driven evader is followed by cars in a certain formation.

The central feature of the entire system is that the supporting middleware, as well as the principles of application development, allow for rapid development and deployment of such scenarios with a multitude of sensors, actuators, computational nodes, over a communication network. In the sequel, we describe the architecture, abstractions, services and principles employed in the system.

III. Component Architecture and Middleware

Middleware is software residing between the operating system and the application, which seeks to facilitate application development and deployment.

The starting point of the design is a *component architecture*. This consists of software modules designed for particular well-defined functionalities. For example, *Kalman-Filter, Predictive-Control, Traffic-Scheduling*, etc., are all examples of components. Components can be developed independently and reused across applications. They can execute on any computer in the computer network. They can also be migrated to a location in the network that is best suited for them, given the information flows in the network as well as the latencies experienced in the network. Failed components can be restarted after capturing their state, thus allowing for reliability. These features allow the development of *self-optimized adaptive systems* of a breed different from traditional parameter-tuning based adaptive control. In particular, they allow run-time optimization of network resources vis-à-vis the control application.

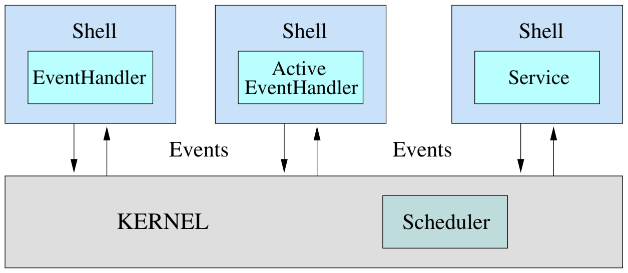

Fig. 2. Architecture of Etherware

To manage components, we have developed a message-oriented middleware, called *Etherware*. It provides interfaces for the creation, upgrade, and migration of components. It also allows components to interact with collocated or remote components through compatible protocols. It allows application developers, for example control engineers, to "address" a component without regard to its actual physical location which, in fact, can even vary during system operation due to its migration. This is the property of "location inde-

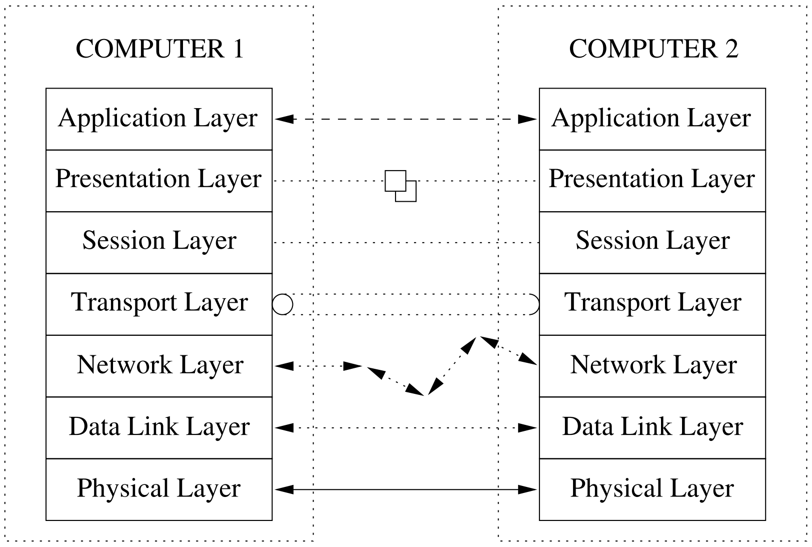

Fig. 3. OSI ISO Network Model

pendence". The middleware has features such as state-capture which are needed for component re-start as well as migration. This state capture is done through the Memento design pattern [7]. Virtual inter-component connections, called MessageStreams, are also maintained across restarts.

The architecture of Etherware is based on the micro-kernel concept [5] as illustrated in Figure 2. Briefly, the *Kernel* represents the system invariant, and is the only part of Etherware that cannot be changed at run-time. It is a very simple and robust entity whose only function is to manage components and deliver messages between them. The Kernel uses a *Scheduler* to schedule message delivery. All other Etherware services are implemented as components. This highly modular architecture support very specific service configurations where unnecessary services can be easily excluded. Further, failures in service components are tolerated, and the services can even be upgraded at run-time.

IV. The Virtual Collocation Abstraction

The architecture of the OSI stack, shown in Figure 3, below features several abstractions, that of "edge," "graph," and "pipe." The data link layer manufactures the abstraction of an "edge," which is used to manufacture the abstraction of a "graph" by the network layer, which in turn is used by the transport layer to manufacture the abstraction of a "pipe." The hierarchical TCP/IP based architecture of the Internet, along with the peer-to-peer protocols, are well matched to these abstractions. The end result is a plug-and-play capability not only for the application, but also for the individual layers themselves. This in turn promotes system longevity by allowing for protocols to be replaced over time in a modular way, while also allowing for modular development. The end result is massive proliferation with over 300 million hosts on the Internet today [6].

The question that arises in our context is: What are the appropriate abstractions for the convergence of control with communications and computing?

We propose an abstraction of *virtual collocation*. To promote proliferation, we believe that it is critical to shrink the development and deployment cycle-time. The critical resource for this phase is the control designer's time.

Therefore our goal is to facilitate control system development and deployment by providing an abstraction to which control design is very well suited. We contend that such an abstraction, taken almost as a default in many electro-mechanical systems where delay is not an issue, is that the distributed control system and plant are actually collocated. For this purpose we need to render invisible to the designer several distracting details. One example is how to address different nodes. In a system where nodes may be added or removed or updated, and one which is always in a state of flux, addressing details should be managed by a "services" layer, and indeed our Etherware middleware does this. It allows the control designer to refer to Kalman-Filter-A as simply *Kalman-Filter-A* without reference to the IP address of the node it is currently located at, and indeed one which may change dynamically under the

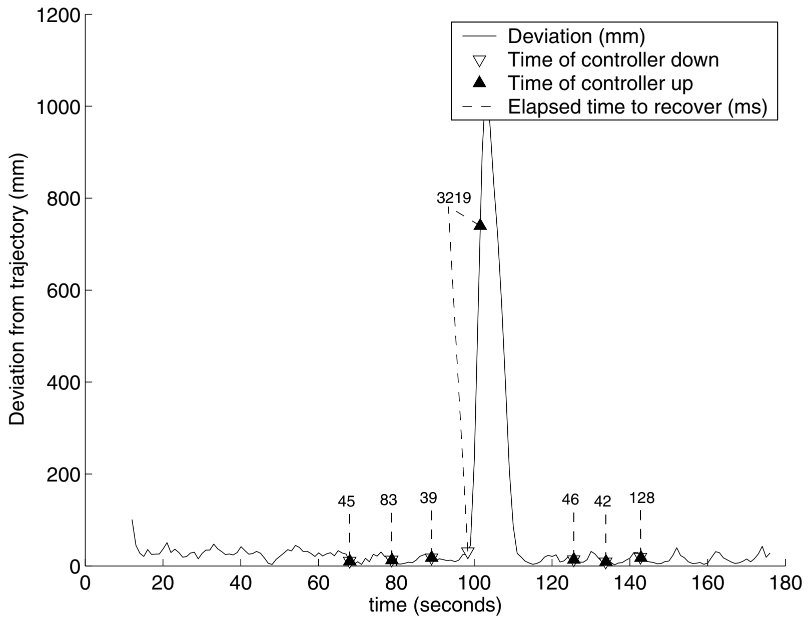

Fig. 4. Car controller restarts in testbed

operation of a self-optimization policy at run-time, in response to packet delays experienced in the network at run-time. Thus the spatially distributed aspects of the system are hidden from the control designer.

Another key feature provided as a service is that of a uniform notion of "time." No two clocks in a distributed system are exactly the same, and we provide a timing translation service [9], which enables any node to regard its own clock as the global clock with all message time stamps automatically translated by services running over the middleware. Thus all information gathered by sensors is automatically time-stamped.

The net result of semantic addressing, location independence of components, and time-translation, is to provide an abstraction of virtual collocation that the control designers can design to. The design of collocated control systems via linear design theories, for example, is very well developed.

A second aspect, also related to time, is that of estimating packet delays experienced over the network. For this, a delay-metering service to provide a profile of current delays being experienced between end-points in the network at run-time is under development.

V. Principle of Local Temporal Autonomy

In a distributed control system it is useful to protect components from the failures of other components. This enhances reliability and robustness, besides making deployment and system evolution easier [9].

We propose a principle of *local temporal autonomy*, whereby the design has as its goal the enabling of a component to be able to function for at least a limited period of time in the face of failures of some other components. During this limited period of autonomy, procedures such as component restart or migration can be invoked to ensure continued uninterrupted of the system.

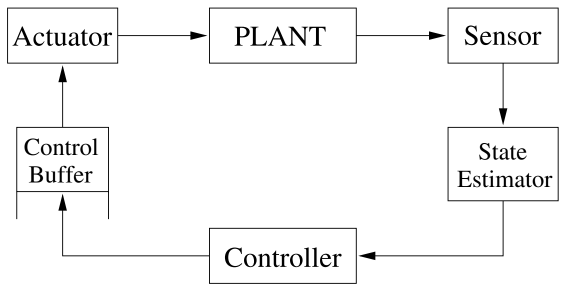

Fig. 5. Local Temporal Autonomy

An example is clarifying. If one directly uses position and orientation estimates provided by an image processing algorithm, then failure of the image processing on the communication network can disrupt the actuation. A solution is to interpose a Kalman Filter as an architectural construct. This can serve as a buffer, and also provide predictions of the future state over a window. These predictions can be repeatedly computed *en bloc* and stored in an actuator buffer, as shown in Figure 5.

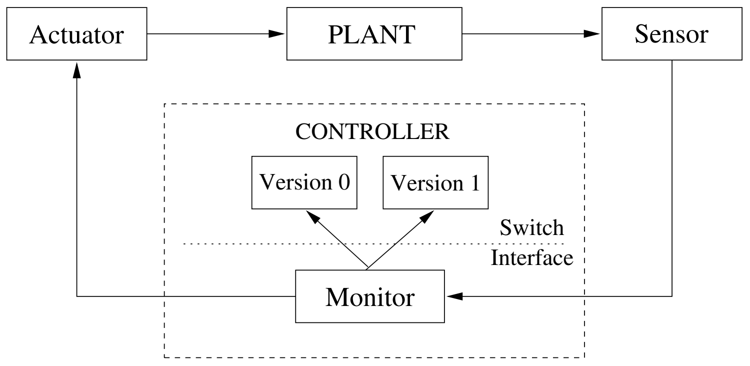

Fig. 6. Simplex based controller design

This allows for temporary disruptions up to the time-window of the prediction and the actuation buffer, thus protecting the actuator against temporary losses of neighboring components.

This window of limited temporal autonomy can also be used for failure recovery. For instance, one of the failure recovery strategies implemented in Etherware is efficient component restarts. To demonstrate this, we conducted an experiment in the testbed where faults were injected into a car-controller at run-time. The deviation in the car trajectory during these restarts is shown in Figure 4.

We see that for six of the seven faults, the controller was efficiently and correctly restarted, and the error in the car trajectory was minimal. This is primarily due to local temporal autonomy of the actuator module, complemented by efficient restart mechanisms in Etherware. As a comparison, during the fourth fault in the experiment, the local application process, including middleware, was restarted. The subsequent large deviation of the car from its trajectory demonstrates the inefficiency of such an approach, and underscores the need for recovery support in middleware.

Indeed the very process of ensuring local temporal autonomy forces a discipline on the control designers to make components as independent as possible. This has the valuable by-product of facilitating other features such as component migration.

VI. Evolutionary Design of Systems

The next aspect we touch on in this tutorial is that of designing to a moving application target. Development of large distributed systems, such as that for traffic control, is often not done with a unifocal goal as a one-time design effort. Rather the application grows as a feature "bloat."

To address this inevitable aspect of design, we suggest an evolutionary approach to design, which is itself designed to enhance reliability at all stages of design. We grow the system by using the Simplex approach of Sha, et. al [10].

In our context it consists of first building a "Monitor" with a "switch" interface, as shown in Figure 6. "Version 0" of a module is first deployed. When a newer version is developed, then during its initial deployment the older, more reliable, but perhaps less performance oriented module is used to monitor it for outlier decisions.

This approach has been successfully used for interposing Traffic Scheduling Algorithms, Collision Avoidance Maneuvers, etc. It can be regarded as a "Design Pattern" [7] for the process of design.

VII. The Need for Theory

We are currently in a situation where "theory" has become a bottleneck. We can deploy functionalities for which unfortunately theoretical understanding and support is lacking. An example to consider is the self-optimization of networked control systems at run-time. In a distributed system, a component for traffic scheduling or filtering can be located at any computer. Given probability distributions of delays across a network, two questions arise: (i) Where should such components be migrated to?, (ii) What control feedback law is optimal in scenarios where the estimator and the actuator are not collocated, so that the estimator only has delayed or even missing samples of actuator commands deployed? Both problems are open. In the 1960s these problems were investigated; see Witsenhausen [11]. While some special situations were found tractable, see Ho and Chu [12], [13], the general outlook was regarded as bleak with regard to obtaining exploit optimal solutions; Witsenhausen [14]. However there is no need for either explicitness or even optimality. Rather, attention can probably be directed to performance improvement oriented solutions for distributed control. This is reminiscent of the mode for algorithm development in computer science. A second approach, more speculatively, is whether one needs an altogether different approach, akin, say, to how economic theory, e.g., the consumption function, has developed; see Keynes [15][1].

VIII. Concluding Remarks

In this tutorial, we have highlighted the importance of abstractions, architecture, middleware, services, and theory for the coming generation of distributed networked control system. We have illustrated the issues on the traffic control testbed Convergence Laboratory at the University of Illinois.

This middleware and services, and the principles themselves can be used in other contexts. The same architecture can be used for building wide temperature control systems by simply replacing code for cars by code for thermostats. The goal is to move to an era of "general purpose control systems" over shared computational and network resources, in much the same way as computational facilities and communication networks are now shared. It is hoped that this approach will lead to a similar proliferation of networked control systems.

References

[1] H. W. Bode, *Network Analysis and Feedback Amplifier Design.* Van Nostrand, 1945.

[2] R. E. Kalman, "Contributions to the theory of optimal control," *Bol. Society Mat. Mexicana*, vol. 5, pp. 102–119, 1960.

[3] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Trans. ASME. (J. Basic Eng.)*, vol. 92D, pp. 34–45, March 1960.

[4] R. E. Kalman, "Canonical structure of linear dynamical systems," *Proceedings of the National Academy of Sciences*, vol. 48, pp. 596–600, 1962.

[5] A. Silberschatz, P. Galvin, and G. Gagne, "Applied Operating System Concepts," John Wiley and Sons Inc, 2000.

[6] Internet Systems Consortium, "ISC Internet Domain Survey", *http://www.isc.org/ds/*

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

[8] S. Graham, G. Baliga, and P. R. Kumar, "Issues in the convergence of control with communication and computing: Proliferation, architecture, design, services, and middleware," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, (Bahamas), pp. 1466–1471, December 14-17, 2004.

[9] S. Graham, G. Baliga, and P. R. Kumar, "Time in general-purpose control systems: The control time protocol and an experimental evaluation," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, (Bahamas), pp. 4004–4009, December 14-17, 2004.

[10] 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.

[11] H. S. Witsenhausen, "On information structures, feedback and causality," *SIAM Journal on Control*, vol. 9, pp. 149–160, 1971.

[12] Y. C. Ho and K. C. Chu, "Team decision theory and information structures in optimal control problems - Parts I and II," vol. AC-17, no. 15-22, pp. 22–28, 1972.

[13] Y. C. Ho and K. C. Chu, "Equivalence of information structures in static and dynamic team problems," vol. AC-18, pp. 187–188, 1973.

[14] H. S. Witsenhausen, "A counterexample in stochastic optimal control," *SIAM Journal on Control*, vol. 6, pp. 131–147, 1968.

[15] J. M. Keynes, *The General Theory of Employment, Interest and Money.* Macmillan Cambridge University Press, 1936.

[1]We are grateful to Pravin Varaiya for this remark.