A Middleware Architecture for Federated Control Systems[1]

Girish Baliga[2]
Dept. of Computer Science and
Coordinated Science Laboratory,
University of Illinois at Urbana-Champaign

P. R. Kumar
Dept. of Electrical & Computer Engineering and
Coordinated Science Laboratory,
University of Illinois at Urbana-Champaign

Abstract

A Federated Control System (FCS) is composed of autonomous entities that cooperate to provide a global behavior. Our focus is on the design of such systems for the convergence of sensing and actuation with communication and computation. A well-designed software architecture is indispensable to orchestrate them. This paper presents the study of one representative system and the ongoing development of a middleware architecture to support software operating it.

I. Introduction

Our focus is on systems composed of entities operating with autonomous control. Examples range from automated production plants to traffic control systems. Traditionally, the software controlling such systems has been tailored to meet the requirements of each individual system. However, this approach of custom, one of a kind design, is costly and not conducive to proliferation. In recent years, technology has evolved to the point where powerful processors embedded in the environment can support generalized approaches. Such approaches can utilize formal theory and well-defined methodologies of design, and can be made available as commercial technologies. However, generalizations have to evolve from experience and be validated in practice.

In this paper, we present a study involving a complex system with multiple autonomous agents and a concomitant middleware architecture that is being developed. Section II provides a brief overview of the prototype and develops the notion of a Federated Control System (FCS). Section III presents a first implementation of the prototype system and considers the issues that arise. Section IV considers related research and technologies that could be used to implement the system. Section V introduces the concept of an Element, which represents a service abstraction for software components in an FCS. Section VI presents a middleware architecture being developed to support the Element abstraction and Section VII concludes.

---

[1] This material is based upon work partially supported by AFOSR under Contract No. F49620-02-1-0217, DARPA under Contract Nos. F33615-01-C-1905 and N00014-01-1-0576, USARO under Contract Nos. DAAD19-00-1-0466 and DAAD19-01010-465, DARPA/AFOSR under Contract No. F49620-02-1-0325, and NSF under Contract No. NSF ANI 02-21357. 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.

[2] Please address all correspondence to the first author. Address: CSL,UIUC, 1308 W. Main St., Urbana, Illinois 61801. Email: gibaliga@uiuc.edu. Web: http://black.csl.uiuc.edu/ ~baliga/

II. A PROTOTYPE SYSTEM

Figure 1 illustrates the prototype system of interest. The system consists of a set of cars moving on a track. Each car is remotely controlled using an RF transmitter, which is in turn connected to the serial port of a dedicated laptop. The speed and direction of a car is controlled by a program executing a control algorithm on its dedicated laptop. The controls are emitted as a stream of alphabetic characters into the serial port, which is connected to a micro-controller that converts this character stream into a sequence of signals to operate the car. The control program that runs the car performs closed loop control and obtains the necessary feedback from an array of cameras (currently two). At intersections, traffic lights illustrate the arbitration of traffic.

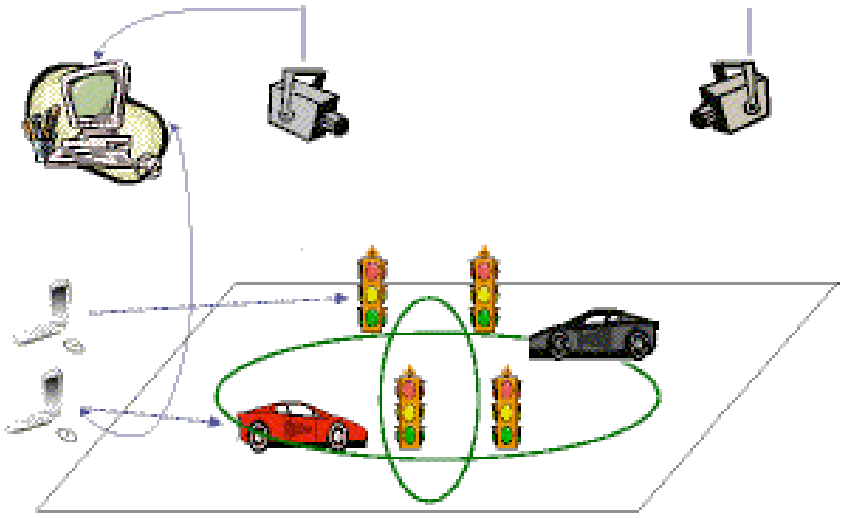

Fig. 1. A Prototype Federated Control System

The system is operated using a set of laptops, each controlling a car. Feedback is provided by a set of desktops that process vision feed, one from each camera. The desktops run programs on Windows and the laptops on Linux. The computers are connected by Ethernet and an 802.11 wireless LAN.

This prototype is meant to be a representative of a class of systems that are composed of sensors, actuators, computers and communication networks that have various control algorithms operating on various parts with strict real-time constraints. Such systems can be categorized as the class of federated control systems defined as follows.

Definition 2.1: A **Federated Control System**[3] (FCS) is a networked system of sensors, actuators and computers that are composed into a set of co-operating entities which operate with autonomous control.

Sensors and actuators represent the interface between the system and its environment. Computers represent programmable components in the system. These components are networked by a topology of connections imposed by system design, technology and convention. Compositions of components represent independent entities in the system, which operate with autonomous control. An example of such a composition in the above system is a car, which is composed of the controlling laptop, the RF connection,

[3] The term "Federated Control System" is adopted from [1].

and the actuators on the car itself. Useful behavior is obtained when these autonomous entities cooperate to achieve desirable goals.

III. A Simple Client-Server Architecture

Our study of the prototype FCS of Section II has paralleled the implementation of a simple working system. The system design emerged as a bottom-up composition of code fragments that were written for simple use-cases [2] of various sub-systems. The overall software architecture of the first working implementation is illustrated in Figure 2. The vision sensor is a camera providing a vision feed to a dedicated desktop that executes an image processing application and determines the position and orientation of cars on the track using a local coordinate system. A central data server connects to the vision sensors (the programs on the desktops), and obtains the vision information for all cars in the system, which it then transforms to a global coordinate system. A central trajectory server computes and serves trajectories for all cars in the system. A trajectory is a sequence of car position coordinates in time. The trajectory server obtains location information about the cars from the data server.

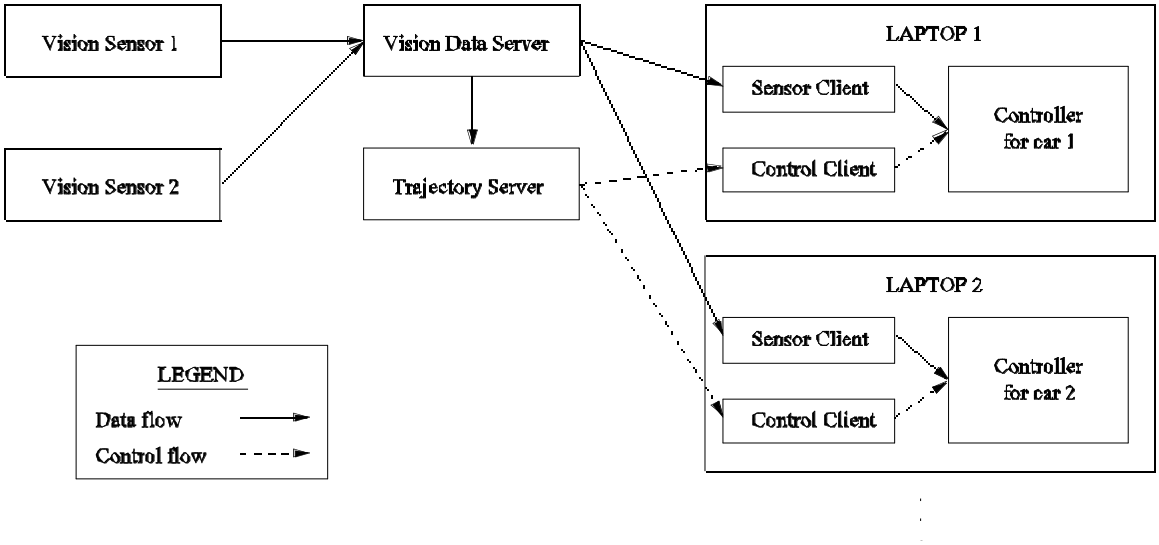

Fig. 2. A simple Client Server Architecture for the prototype system

Each laptop executes a control algorithm that moves the car according to a trajectory provided by a trajectory server. The control program obtains feedback from the data server. This implementation is working quite well. We have tested scenarios with up to 8 cars operating simultaneously on the track and closely following pre-specified trajectories. However, some fundamental problems were encountered during the system implementation.

- **Location Independence:** Considerable pre-configuration was necessary to operate the system. For example, all the clients needed to know addresses and ports of the respective servers. Moving the servers to different machines entailed much reconfiguration.
- **Bootstrap:** Starting up the system was an elaborate process. The servers and clients needed to be started in the right order and on pre-specified computers. Synchronization (implemented at the trajectory server) was needed to ensure that all the cars were ready before any car started to move.
- **Software update:** Any change to the car control program had to be propagated to all the laptops, requiring a system-wide reboot.
- **Software model and architecture:** The programming was done concurrently by multiple members of the team working on the different problems of image processing, car control, car scheduling [3] and infrastructure. In particular, it was quite a challenge to separate the infrastructure code as an independent body. Maintenance and update of this code required a lot of change and testing for the other code using it.

The next iteration of the system implementation is an ongoing attempt to solve these and other problems. In what follows, we outline the ongoing research in this direction.

IV. Related research and applicability

Some of the problems considered in Section III, such as location independence and pre-configuration have elegant solutions in commercial middleware technologies such as CORBA [4], DCOM [5] and JINI [6]. However, there are some fundamental problems in using these as the base infrastructure. One key problem is that these middleware technologies are intended for transaction-based applications, where the focus is on providing a rich interface. However, FCSs requires an event-based infrastructure where every update of data or control can proceed asynchronously.

OCP [7] extends CORBA to support event streams and other primitives. However, there are still some unsupported features. For example, the problem of software update is not addressed in the infrastructure and needs to be engineered into the system. Besides, OCP is essentially a transaction-based system with relevant functionality imposed as a new layer.

Message-Oriented Middleware such as IBM's WebSphere MQ [8], Sun's Java Message Service [9] and Microsoft's Message Queue Server [10] represent the commercial event based middleware technologies. However, these middleware platforms focus on business applications where secure delivery of data is more important than timeliness and such other requirements of FCSs.

Some other interesting approaches to providing software infrastructure for control systems include Cactus [11] and Giotto [12]. Control frameworks such as AOCS [13], Realtime Framework [14] and NEXUS [15] have interesting approaches to specific classes of systems. [16] provides a good overview of research and technology that has been developed for FCSs and related domains.

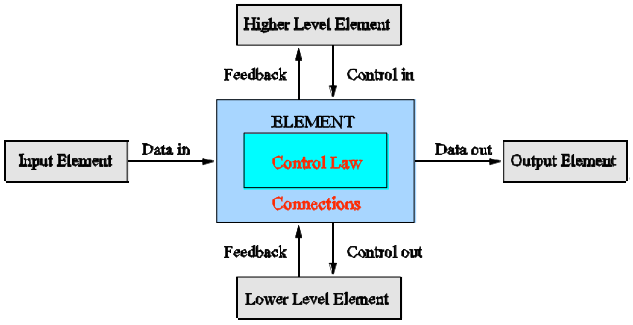

Fig. 3(a). The Element abstraction in FCSs.

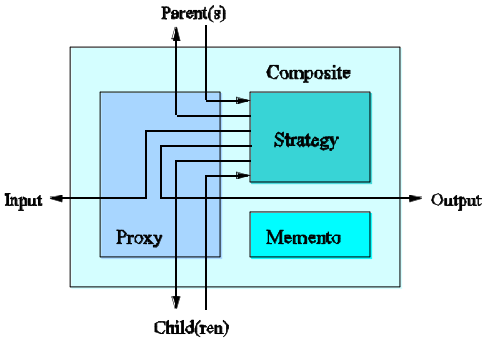

Fig. 3(b). The Element pattern

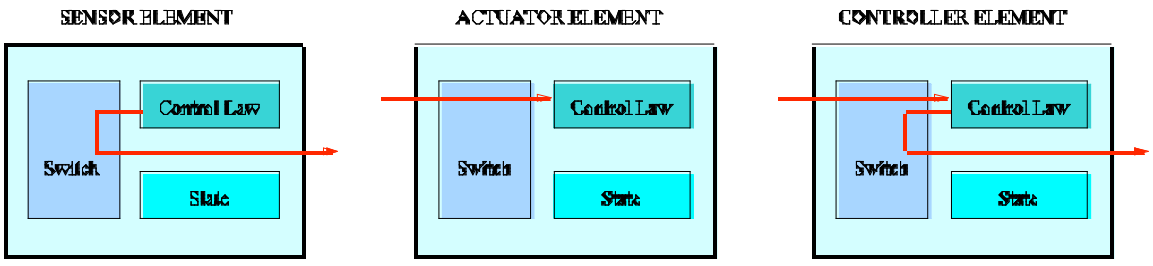

Fig. 3(c). The Elements in an FCS

V. The "Element" pattern

The brief survey in Section IV shows that middleware is one of the predominant architectural models that have been applied to FCSs. Middleware provides a nice separation of infrastructure from rest of the software. However, a good service model is necessary for software components to be designed, implemented and operated on the infrastructure layer. Figure 3(a) presents one candidate model, called the "Element" abstraction that models software components in an FCS.

An Element encapsulates a control algorithm operating on some sub-system in the FCS. An Element participates in a control hierarchy. For example, in the architecture of Figure 2, if car control were a representative Element, then the trajectory server would be a "higher level" Element, and the car actuator would be a "lower level" Element. An Element also participates in a data flow. In Figure 2, the data server would be an "input" Element to car control.

However, not all Elements need have all the connections shown in Figure 3(a). Figure 3(c) for example shows connections that would exist for sensors, actuators and controllers in an FCS. The Element abstraction can be represented in many ways. The following development follows [17] and represents the abstraction as a design pattern. As Figure 3(b) shows, the Element pattern is essentially a composition of patterns from [17].

- A **strategy** represents the control law implemented by the element.
- A **memento** corresponds to the concept of state in a control law.
- The **proxy** operates as the conduit for communicating with other Elements in the system.
- The **composite** pattern denotes the idea that multiple Elements can be handled as a collection.

Strategy and Memento represent the service model for Elements in an FCS. Proxy is the interface for an Element to the rest of the system and is analogous to Skeletons and Stubs in CORBA [4]. The composite pattern represents how the infrastructure would view the set of Elements in the system.

Connections to other Elements are modeled as streams of asynchronous events. This represents the common case where an Element gets periodic updates from a left Element or a sequence of control points from an upper Element. Synchronous communication such as a handshake may occur during startup of connections between Elements. This can be represented by a pair of asynchronous event streams hidden by the proxy. The thesis [18] presents further details of the Element abstraction and considers many other aspects of the behavior of Elements.

We conclude by briefly considering how this architecture addresses the issues raised in Section III. Location independence is obtained by requiring Elements to have addressable profiles, which are then used to identify and address them. Elements are required to have a well-defined state (communicated as mementos [17]) that can be used to pause and restart them. This permits bootstrapping and update of Elements by downloading code and initial state from one central location. The middleware architecture provides also provides a clean separation of application code from the infrastructure.

VI. An Element Based Architecture

One approach to providing infrastructure for Elements in an FCS is to design a simple architecture that directly supports Elements as the primitive components. Figure 4 shows an architecture for Element based systems. Each Element communicates with its shell. The shell represents the Element to the system, and vice versa. It also provides a convenient mechanism to maintain Element specific data structures and modifications for additional functionality (for example, adaptation for synchronous communication).

The Elements are managed as a collection (the composite pattern [17]) of similar entities by the manager. Since each Element would potentially operate in its own thread, the manager schedules the threads and supports various policies. Similarly, the events that flow between Elements are also scheduled. The events are maintained in an Event Board, which is accessed uniformly by the Element shells and the rest of the system.

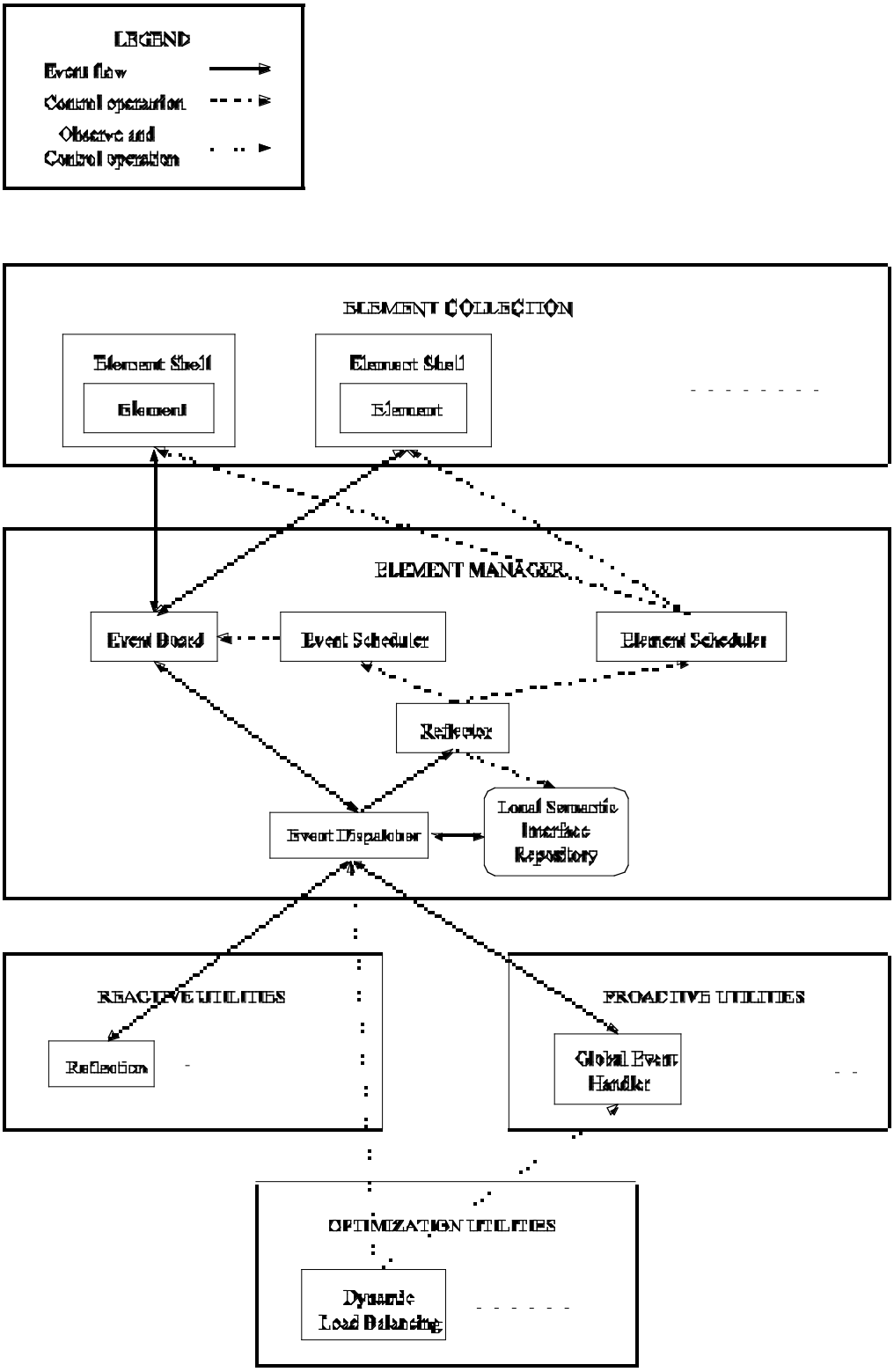

Fig. 4. An architecture for supporting the Element abstraction

In addition to the manager, three sets of utilities are provided.

1. **Reactive utilities** are passive components that are activated by appropriate events requesting their services. An example of such a service is the "reflector". This service consumes the reflection [19] events generated by the elements and exposes the system infrastructure as meta-data.
2. **Proactive utilities** are active components that generate events as well as consume them. One such component is the global event dispatcher that forwards events generated by the local elements addressed to remote elements and vice versa.
3. **Optimization utilities** are components that do not participate directly in event processing activities. Instead, they observe other components (through notification mechanisms for example) and perform optimizations as appropriate.

The thesis [18] considers the rationale, functions and responsibilities of the various components of this architecture in more detail.

We conclude by noting that the notions of Elements and Element architectures have been further developed to EtherArch – a software architecture for FCSs. This work is currently under review for publication.

VII. Conclusions

We have presented the design of middleware architecture for federated control systems. The notion of an FCS was elaborated through the description of a prototype system. Some interesting problems that were encountered while developing software to operate such systems were motivated by a first implementation. These problems were addressed by the concept of an Element, a design pattern to encapsulate it, and a middleware architecture to support it. Current research addresses the implementation of this architecture and a refinement and formalization of the Element abstraction.

References

[1] J. S. Bayne, "Automation and control in Large-Scale interactive systems," in *Proceedings of ISORC*, April-May 2002, pp. 3–12.
[2] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 5th ed. McGraw-Hill, November 2001.
[3] A. Giridhar, "Scheduling traffic on a road network," Master's thesis, University of Illinois at Urbana-Champaign, December 2002.
[4] *Common Object Request Broker Architecture: Core Specification*, Object Management Group Inc, November 2002.
[5] *DCOM Technical Overview*, Microsoft Corporation, November 1996.
[6] K. W. Edwards, *Core JINI*, ser. The Sun Microsystems Press. Upper Saddle River, NJ 07458: Prentice Hall PTR, 1999.
[7] L. Wills, S. Sander, S. Kannan, A. Kahn, J. V. R. Prasad, and D. Schrage, "An open control platform for reconfigurable, distributed, hierarchical control systems," in *Proceedings of the Digital Avionics Systems Conference*, October 2000.
[8] P. R. Jenkins, *WebSphere - A "What Is It?" White Paper*, Candle Corp., El Segundo, Ca.
[9] K. Haase, *Java Message Service API Tutorial*, Sun Microsystems, Inc, Palo Alto, Ca, 2002.
[10] P. Houston, *Building Distributed Applications with Message Queuing Middleware*, Microsoft Corp., Redmond, Va, 1998.
[11] M. Hiltunen and R. Schlichting, "The cactus approach to building configurable middleware services," in *Proc. Workshop Dependable System Middleware and Group Communication (DSMGC 2000)*, Nuremberg, Germany.
[12] T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: A time-triggered language for embedded programming," in *Proc. 1st Int. Workshop Embedded Software (DMSOFT '01)*, Tahoe City, Ca, 2001.
[13] T. Brown, A. Pesetti, W. Pree, T.Henzinger, and C.Kirsch, "A reusable and platform-independent framework for distributed control systems," in *Proc. IEEE/AIAA 20th Digital Avionics System Conference (DASC'2001)*, Daytona, Fl, October 2001.
[14] A. Traub and R. Schraft, "An object-oriented realtime framework for distributed control systems," in *Proc. IEEE Conference on Robotics and Automation*, Detroit, Mi, May 1999.
[15] J. Fernandez and J. Gonzalez, "Nexus: A flexible, efficient and robust framework for integrating software components of a robotic system," in *Proc. IEEE ICRA*, 1998.
[16] B. S. Heck, L. M. Wills, and G. J. Vachtsevanos, "Software technology for implementing reusable, distributed control systems," *IEEE Control Systems Magazine*, vol. 23, no. 1, February 2003.
[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns*, ser. Professional Computing Series. Addison-Wesley, 1995.
[18] G. Baliga, "A software architecture for federated control systems," Master's thesis, University of Illinois at Urbana-Champaign, December 2002.
[19] F. Kon, F. Costa, G. Blair, and R. H. Campbell, "The case for reflective middleware," *Communications of the ACM*, vol. 45, no. 6, June 2002.