AN OPEN CONTROL PLATFORM FOR RECONFIGURABLE, DISTRIBUTED, HIERARCHICAL CONTROL SYSTEMS

Linda Wills,[1] Sam Sander,[1] Suresh Kannan,[2] Aaron Kahn,[2] J.V.R. Prasad,[2] Daniel Schrage[2]

[1]School of Electrical & Computer Engineering
[2]School of Aerospace Engineering
Georgia Institute of Technology, Atlanta, GA 30332

Abstract

Complex control systems for autonomous vehicles require integrating new control algorithms with a variety of different component technologies and resources. These components are often supported on different types of hardware platforms and operating systems and often must interact in a distributed environment (e.g., in communication with a groundstation, mothership, or other UAVs in a swarm). At the same time, the configuration and integration of components must be flexible enough to allow rapid online reconfiguration and adaptation to react to environmental changes and respond to unpredictable events during flight, such as avoiding a moving obstacle or recovering from vehicle equipment failures. This paper describes an open software architecture, called the Open Control Platform (OCP), for integrating control technologies and resources. The specific driving application is supporting autonomous control of vertical take-off and landing (VTOL) uninhabited autonomous vehicles (UAVs).

Introduction

Achieving autonomous flight for high agility, extreme performance aerial vehicles poses significant challenges to current control systems engineering. It requires integrating innovative control algorithms with a variety of component technologies and resources, including advanced perception and sensor processing, simulation and visualization, geographic information systems, and global coordination resources. The challenge is to deal with the complexity of integrating these components in a distributed environment, while still providing the flexibility to reconfigure them dynamically. This paper describes an open software architecture, called the Open Control Platform (OCP), for integrating control technologies and resources, which is being developed as part of the DARPA Software Enabled Control Program. The specific driving application is supporting autonomous control of vertical take-off and landing (VTOL) uninhabited autonomous vehicles (UAVs) [1]. The OCP extends current real-time distributed object computing technology to coordinate distributed interaction among hierarchically organized components and to support dynamic reconfiguration of the components. The OCP allows systems of components with standardized interfaces to be connected while abstracting away implementation details of lower level components. This enables components to be easily switched, rewired, and adapted at run-time to support dynamic reconfiguration.

This paper first gives background on the UAV application context and typical control system architectures that need to be supported. It then describes the control-specific application programmer interface (Controls API) to the OCP and the process of using the OCP. Working examples from the VTOL UAV application are highlighted. These include results of a telemetry flight test in which the OCP was used to perform a smooth online switch between two sensors providing telemetry data at different rates. Finally, open research issues and future directions are described.

Challenges of Integrating Diverse, Distributed Control Components

The OCP is being developed to support the integration and rapid run-time reconfiguration of complex control systems. The particular application chosen to drive OCP development is autonomous control systems for extreme-performance UAVs, such as the Yamaha R-50/R-max helicopter and the X-cell VTOL UAV. Autonomous flight requires dealing with unexpected external threats, such as moving obstacles, and unpredictable internal failures, such as a tail rotor failure, that can disrupt a mission.

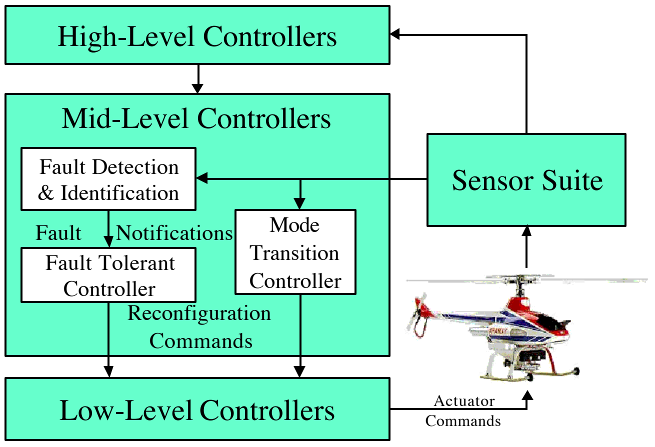

Figure 1. Control Hierarchy

A typical way of structuring the UAV control system is shown in

Figure 1. It uses a hierarchy of control algorithms [2]:

- Low-level control algorithms in the flight control system perform stability and control augmentation [3,4,5,6,7]
- Mid-level control components respond to discrete events, such as a mode change or the occurrence of a vehicle failure. Mode transitioning manages the graceful transition form one operational mode (such as hover) to another (e.g., fly forward) [8]. Other mid-level control components perform fault detection, identification, and fault-tolerant control reconfiguration to recover from vehicle failures.
- High-level control components perform mission planning and replanning and choose a sequence of operational modes to execute the mission plan [9]. It does this based on an assessment of progress toward the mission goals and awareness of external obstacles and the location of the target.

The hierarchy of control system layers helps control the complexity of differing time scales among components and the interaction between components with continuous dynamics at the lowest level and discrete-event components at the highest level. Mid-level components provide a bridge between the two, for example, by managing a continuous, smooth transition from one discrete mode to another.

Developing hierarchical control systems like these involves integrating many different types of components, including software algorithms, device drivers for hardware components, sensor data processing code, and mathematical flight dynamics models written in more than one programming language. The components may also originate from a variety of sources: newly developed using Matlab/Simulink, off the shelf commercial products, or legacy components. These systems are further complicated in that the components are often distributed across different hardware platforms and types of networks. In the UAV application, components running on the on-board computer may communicate with monitoring components on groundstation computers. They may share navigational or proximity data with other UAVS in a swarm and they may receive mission planning information from the swarm's coordinating mothership.

Integrating these diverse components so that they seamlessly communicate with one another both locally and remotely is challenging. Building these systems requires detailed, tedious activities: complex timing relationships have to be worked out, data interchange protocols have to be created and maintained, detailed network programming (e.g., using sockets and remote procedure calls) has to be performed, and careful prioritization and scheduling of tasks has to be accomplished.

Even more challenging is trying to *change* these systems – either offline (e.g., to incorporate new types of sensor technology) or at runtime (e.g., to switch to a new control algorithm in response to a vehicle failure). Yet, the ability to flexibly and easily adapt the system architecture is critical to the ability to reuse and to the "plug-and-play" extensibility of complex control systems. Furthermore, rapid adaptation is critical to enabling innovative online customizable controls technology for extreme-performance applications, such as aerial robotics.

Role of the OCP

The primary motivation for developing the Open Control Platform is to raise the conceptual level at which the controls engineer integrates components of complex, distributed, and reconfigurable control systems. Rather than programming at the level of network protocols, sockets, memory byte orderings, and remote procedure calls, the OCP supports flexible component integration, providing an abstract interface based on familiar controls engineering concepts, such as block diagram components, input and output ports, and signals (e.g., measurement or command signals). As shown in Figure 2, the OCP consists of a core real-time distributed computing substrate, wrapped with a Controls API layer that provides abstractions that bridge the gap between the controls domain and the core distribution substrate. Our research focuses on development of the Controls API layer. For the core OCP, we experimented with two different distributed communication substrates. We designed the Controls API to allow different cores to be plugged in as new distribution substrates become available or improvements are made to the existing technology in the future.

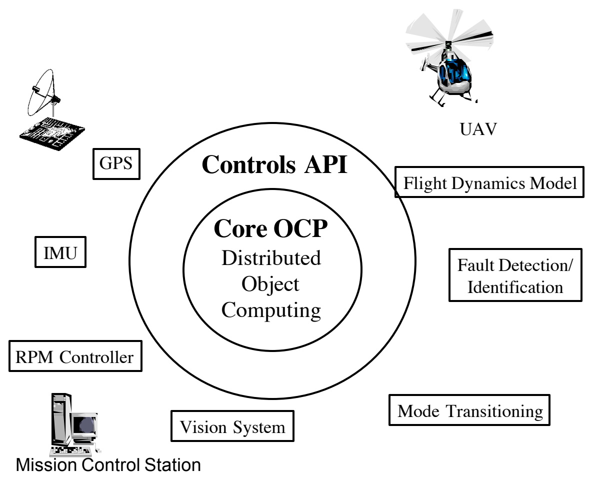

Figure 2. OCP Integrates Distributed Components Using Controls Domain Abstractions

Core OCP

Recent advances in distributed object computing have provided middleware technology to allow distributed, heterogeneous components to interoperate across diverse platforms and network protocols in real-time. Specifically, we have successfully leveraged from two different distribution substrates. One is being developed by Boeing Phantom Works and Washington University and is based on Real-Time CORBA [10]. CORBA, which stands for Common Object Request Broker Architecture, is a standard set by the Object Management Group (OMG) [11] to achieve seamless distributed communication between objects running on different computers and across multiple network protocols. Real-time CORBA extends CORBA technology to allow distributed communication to occur in real-time [12,13]. The other distribution substrate we experimented with as the core OCP is Real-Time Innovations' Network Data Delivery Service (NDDS) [14]. This section describes the particular mechanisms that these substrates provide which are critical to supporting reconfigurable, distributed control applications.

Our UAV control application imposed three key requirements on the distributed communication substrate formed by the core OCP:

- Plug-and-play extensibility and evolution calls for *decoupled, mediated communication* between components. While tightly coupled components can make the system faster, it makes changes to the system extremely difficult to achieve, particularly online. It severely reduces the ability to interchange components on the fly.
- *High-performance distributed communication* is needed to ensure that time, memory, and bandwidth resources are conserved, particularly in this application where onboard computational resources are limited and remote communication may occur over wireless links.
- *Online adaptibility* of the control application requires that the distribution mechanism can be dynamically customized and reconfigured at runtime. This requires dynamic scheduling and dynamic resource management.

Decoupled, Mediated Communication

To satisfy the requirement of decoupled, mediated communication, both core OCP distribution substrates support a "publish-subscribe" model of communication. In Real-Time CORBA, this is provided by the real-time event service [12] and NDDS supports what they call RTPS, or the "Real-Time Publish-Subscribe" model [14]. In the publish-subscribe model, a bus-like communication abstraction is used, often called an event channel. Components that generate data ("suppliers") or use data ("consumers") connect to the event channel and the suppliers "publish" certain data event types while the consumers "subscribe" to certain event types. Quality of service (QoS) properties can be associated with subscriptions to specify the real-time prioritization and timing requirements of the events.

The event channel acts as a Mediator [15] between components so that their interconnections are flexible. When a component subscribes to some type of event, (e.g., navigational state information) it does not necessarily care where that information is coming from. It would like to receive that information from whatever sensors or sensor processing modules are active. The software should not make rigid commitments to which components are providing this information by hardcoding a call to a specific source of data. The event channel provides the level of abstraction needed by mediating information flow between suppliers and consumers. For example, when a new type of sensor is added to the system or replaces another type of sensor, it can be connected to the event channel to publish its type of data and all consumers subscribed to that type will receive it.

The system architecture is reconfigured by a relatively local change in connections to the event channel, rather than by more pervasive changes between the suppliers involved and all possible consumers. Thus, the event channel helps to minimize the architectural impact of switching components by localizing the changes needed so that they can be made quickly and with high reliability.

High-Performance Distribution

The primary mechanism used in the core OCP to satisfy the need for high-performance distributed communication is *replication.* This is a common technique in distributed computing which caches local copies of remote data objects so that they can be efficiently accessed frequently by several local consumers.

Online Adaptibility and Reconfiguration

The third requirement of supporting online adaptation of the system means that the mechanisms used to support the other two requirements themselves be customizable at runtime. For example, runtime changes to quality of service parameters and event subscriptions must be supported. Similarly, replication update strategies must be customizable. For example, replicated information being received on the ground may need to be updated more often when a critical mission replanning task is being performed to avoid a threat.

NDDS and Real-Time CORBA differ in their ability to support this dynamic aspect. Briefly, RT CORBA provides more generality and flexibility and more powerful event processing abstractions, such as event correlation and pluggable scheduling strategies. NDDS, on the other hand, provides higher performance and online-customizable replication mechanisms. Boeing is currently developing a Core OCP that uses RT CORBA and incorporates recent advances in dynamic scheduling from Washington University [16] and dynamic resource management from Honeywell Technologies [17, 18].

An important feature of the Controls API we are generating at Georgia Tech is that it makes the underlying distribution mechanisms used by the Core OCP transparent to the controls system developer. This will enable a system designer to easily incorporate improvements to real-time communication technologies as they become available in the future.

Controls API

The Core OCP abstracts away details of dealing with remote objects, network protocols, etc., but it still requires extensive computer science background (e.g., details about event channels, scheduling, and replication) to use it effectively. The Controls API raises the level of abstraction higher to provide a convenient interface familiar to controls engineers, bridging the gap between the controls domain and the core OCP.

A control system is conceptually made up of components, which communicate via signals (see Figure 3). The Controls API provides a component interface for sending and receiving data (e.g., measurement signals and command signals), sending and receiving notifications, and performing system reconfiguration. Notifications include timeout events, signal update events, and correlation events (which are user-specified unions of events).

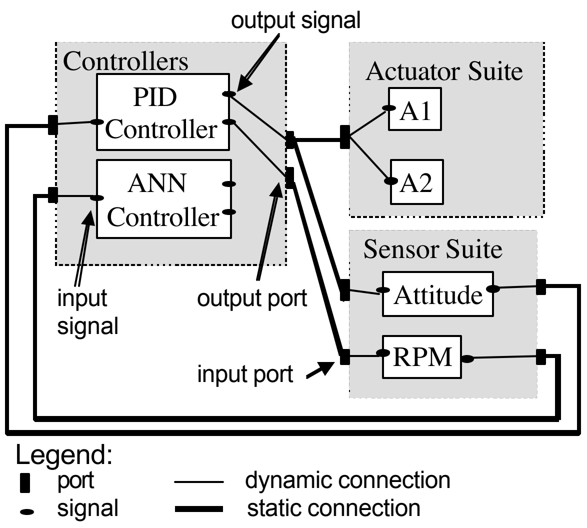

Figure 3. An Example OCP Component Configuration with Ports and Signals

Components (the grey boxes in Figure 3) are collections of subcomponents, each of which consists of user-defined application code. Each subcomponent consumes zero or more input signals and generates zero or more output signals. For example, in Figure 3, the PID Controller receives an attitude measurement signal and it generates a command signal. Components provide a static interface through input and output ports which connect to particular signals of subcomponents – in essence making certain signals visible to other components. The static port interface provides a contract stating which data signals the component consumes or generates, while hiding details about which subcomponent(s) within the component actually use or produce those signals. This facilitates dynamic reconfiguration by allowing subcomponents to be easily replaced by others within a component via a local rewiring of signals to component input and output ports.

Ports represent mediated connection points which contain quality of service information. They can be viewed as the "hinge points" at which reconfiguration can take place by rewiring the signals that are connected to them. They allow subcomponents to communicate input and output signals to one another *anonymously*. The individual subcomponents do not connect directly to one another, but instead they are decoupled. Their communication is mediated through ports and signals, which is maintained via a signal manager. The signal manager hides details concerning the event service and data replication.

Ports and signals are used to transfer data between user-defined components. The data transferred may be aggregate in that it may contain multiple elements (e.g., a measurement signal may be a vehicle state vector).

At runtime, the connections from ports to signals and vice versa can be created, disconnected and reconnected. Ports and signals allow both the application code and the underlying communications mechanisms to have fixed endpoints, while providing for runtime reconfiguration. Ports are the static entity for the infrastructure; while signals are the static entity for application code. The following constraints govern the use of ports and signals in component integration:

- At startup, all ports and port-to-port connections must be setup, and they cannot be changed once they are declared.
- Signals can be declared and connected to existing ports at any time during execution. A signal may also be disconnected from a port at any time.
- Ports may fanout, but not fanin. That is, an output port can connect to any number of input ports, but an input port can be connected to at most one output port.
- Signals may fanout, but not fanin. That is, an output signal may connect to multiple output ports, but each output port can connect to at most one output signal. Similarly, an input port can connect to any

number of input signals, but each input signal can connect to at most one input port.

Application Examples and Results

Flight trials of the OCP were conducted at Georgia Tech using an X-cell helicopter testbed, shown in Figure 4. The goal of the flight trials was to test the dynamic reconfiguration and distributed communication features of the OCP. The basic mission of the X-cell was to obtain telemetry data and communicated it to the ground computer in real time, at rates that could be varied by the user on the ground.

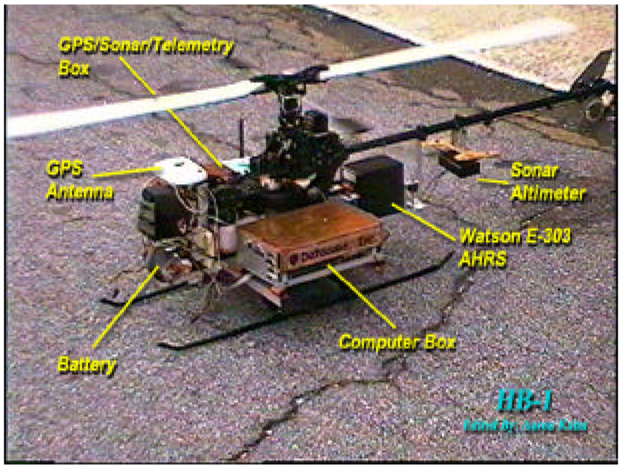

Figure 4. Flight Test Aircraft

The avionics system used in the flight trial consists of mostly off-the-shelf hardware and software components. It uses a 486 flight control computer, running the Linux operating system. The sensor suite consists of an attitude and heading reference system, a sonar altitude sensor, and a Novatel differential GPS. The avionics box is made to be highly modular so it can be ported to other helicopter testbeds. The integration of the hardware and software components for the flight test was performed in a few weeks using the OCP – a significant improvement over previous systems integration efforts which took several months without the aid of the OCP.

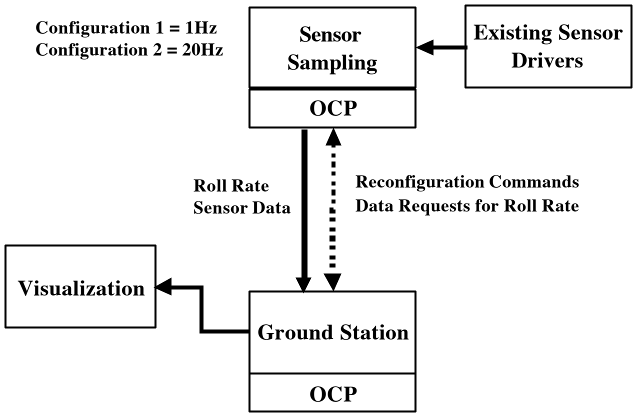

Figure 5. Telemetry Flight Test Data Flow

The block diagram shown in Figure 5 illustrates the communication paths between the primary components. Existing sensor drivers were used to read the sensor data and the data was made available to the OCP application running on board the helicopter. This data was sampled at different rates to represent the realistic situation in which data may be required at different sampling rates by different components. The roll rate sensor data was selected for this application. The sensor output was received on the ground at user-specified rates. The software configuration for the sampling rate was controlled from the ground. That is, commands to switch to a different rate were given from the ground, which caused an online reconfiguration of the software running onboard the vehicle. A video of this flight test can be viewed at www.uav.ae.gatech.edu/sec/.

We have also employed the OCP's integration and dynamic reconfiguration capabilities in the support of innovative mid-level control algorithms for mode transitioning and for fault detection and identification [19, 20]. We have successfully tested these in simulation (see www.uav.ae.gatech.edu/sec for video clips of these tests). We plan to perform hardware-in-the-loop simulation tests, followed by actual flight tests of these mid-level control algorithms on a helicopter testbed in the near future.

Future Directions

The OCP simplifies the integration of complex, distributed control systems while providing the flexibility for rapid online adaptation and reconfiguration. There are a number of areas of ongoing work, including the following.

Allowing runtime changes to quality of service properties of signals is a challenging open issue which will benefit from new dynamic scheduling algorithms [16] and dynamic resource management [17, 18] capabilities which Boeing is incorporating into the core OCP.

A generic interface to mid-level control components is being developed, which will provide support for hybrid control systems. This will provide common reconfiguration primitives used in fault tolerant control reconfiguration, including support for making gradual transitions from one component to another, based on user-defined transition functions. This is particularly useful in hybrid control applications where a discrete switch from one signal to another or a discrete replacement of one component with another is not desirable.

Mechanisms for configuration and reconfiguration validation are also being developed. In addition, tool integration of the OCP with common tools used by controls engineers, such as Matlab/Simulink and RTI's ControlShell, is planned.

As controls applications become more complex and more distributed, new ways of developing and integrating control algorithms are needed. The OCP leverages from recent advances in software technology to enable rapid development, evolution, and online adaptation of these systems.

Acknowledgements

This work is supported under the DARPA Software Enabled Control Program under contracts #33615-98-C-1341 and #33615-99-C-1500, managed by the Air Force Research Laboratory (AFRL). We gratefully acknowledge DARPA's Software Enabled Control program, AFRL, and Boeing Phantom Works for their continued support. We have benefited greatly from the guidance of Helen Gill (DARPA) and Bill Koenig (AFRL). We would also like to acknowledge the contributions of Brian Mendel, Jim Paunicka, David Corman, and Bryan Doerr of the Boeing Phantom Works, and Scott Clements, Murat Guler, Bonnie Heck, Freeman Rufus, Gideon Saroufiem, George Vachtsevanos, and Ilkay Yavrucuk of Georgia Tech.

References

[1] D. Schrage and G. Vachtsevanos "Software-Enabled Control for Intelligent UAVs", *Proc. of 1999 Int. Conference on Control Applications*, Hawaii, August 22-27, 1999.

[2] G. Vachtsevanos, "Hierarchical Control," *Handbook of Fuzzy Computation*, Editors in Chief E. Ruspini, P. Bonissone and W. Pedrycz, Institute of Physics Publishing, pp. F 2.2:42-53, 1998.

[3] B. S. Kim and A. J. Calise. Nonlinear flight control using neural networks. In *AIAA Journal of Guidance, Control, and Dynamics*, volume 20-1, pages 26-33, 1997.

[4] J.V.R. Prasad and A.M. Lipp. Synthesis of a Helicopter Nonlinear Flight Controller Using Approximate Model Inversion. *Mathematical and Computer Modelling* (18)3-4, pp. 89-100, Aug. 1993.

[5] J.V.R. Prasad, A. Calise, Y. Pei, and J. Corban. Adaptive Nonlinear Controller Synthesis and Flight Test Evaluation on an Unmanned Helicopter. *Proc. of the IEEE Conference on Control Applications*, Hawaii, August 1999.

[6] J.V.R. Prasad and I. Yavrucuk. Reconfigurable Flight Control using RPM Control for Heli-UAVs. *Proc. of the 25th European Rotorcraft Forum*, Rome, Italy, Sept. 1999.

[7] R. Rysdyk and A. Calise. Adaptive Model Inversion Flight Control for Tiltrotor Aircraft. *Journal of Guidance, Control, and Dynamics,* vol. 22, no. 3, pp.402-407, May-June 1999.

[8] F. Rufus, G. Vachtsevanos. Flight stability of mode-to-mode fuzzy controllers. *Journal of Guidance, Control, and Dynamics,* vol.22, no.6, pp.823-32, Nov.-Dec. 1999.

[9] G. Vachtsevanos, W. Kim, S. Al-Hasan, F. Rufus, M. Simon, D. Schrage and J.V.R. Prasad, Mission Planning and Flight control: Meeting the

Challenge with Intelligent Techniques, *Journal of Advanced Computational Intelligence*, (1)1:62-70, Oct. 1997.

[10] D. Schmidt and F. Kuhns. An Overview of the Real-Time CORBA Specification, *IEEE Computer*, vol. 33, no. 6, pp. 56-63, June 2000.

[11] Object Management Group. CORBA 2.2 Common Object Services Specification, Dec. 1998. http://www.omg.org.

[12] T. Harrison, C. O'Ryan, D. Levine and D. Schmidt. The Design and Performance of a Real-time CORBA Event Service, IEEE Journal on Selected Areas in Communications, 1999.

[13] D. Levine, S. Mungee, and D. Schmidt. The Design and Performance of Real-time Object Request Brokers. In *Computer Communications*, volume 21, 1998.

[14] G. Pardo-Castellote, S. Schneider, and M. Hamilton. NDDS: The Real-Time Publish-Subscribe Network, white paper, Aug. 1999. http://www.rti.com/products/ndds/ndwp0899.pdf

[15] R. Johnson E. Gamma, R. Helm and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1998.

[16] D. Levine, C. Gill, D. Schmidt. Dynamic Scheduling Strategies for Avionics Mission Computing. In *Proc. 17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference*, pp. C15/1-8, volume 1, 1998.

[17] M. Cardei, I. Cardei, R. Jha, A. Pavan. Hierarchical feedback adaptation for real time sensor-based distributed systems. In *Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000),* pp.181-188, 2000.

[18] B. Doerr, T. Venturella, R. Jha, C. Gill, D. Schmidt. Adaptive scheduling for real-time, embedded information systems. *Proceedings of the 18th Digital Avionics Systems Conference*. p.2.D.5/9 St. Louis, MO. Oct. 1999.

[19] S. Kannan, C. Restrepo, I. Yavrucuk, L. Wills, D. Schrage, J.V.R. Prasad. Control Algorithm and Flight Simulation Integration Using the Open Control Platform for Unmanned Aerial Vehicles. *Proc. of the Digital Avionics Conference*, pp. 6.A.3-1 to 6.A.3-10, St. Louis, MO, Oct. 1999.

[20] F. Rufus, S. Clements, S. Sander, B. Heck, L. Wills, and G. Vachtsevanos. “Software-Enabled Control Technologies for Autonomous Aerial Vehicles,” *Proceedings of the Digital Avionics Systems Conference*, pp. 6.A.5-1 to 6.A.5-8, St. Louis, MO, October 1999.