Chapter 11
Software Technology for Implementing Reusable, Distributed Control Systems*

Bonnie S. Heck, Linda M. Wills and George J. Vachtsevanos

Control systems such as those used in satellites, spacecraft, automobiles, chemical processing plants, and manufacturing rely heavily on the software that is used to implement them. In fact, engineers at Boeing Co. and Honeywell Inc. have estimated that 60–80% of the development of a complex control system is the software development effort, while only 20–40% is the actual control system design. The software effort includes programming individual software modules as well as writing code for communications between components. Much of the software development time is spent on making the software stable and reliable, such as tracing possible hazard conditions and inserting software fault protection methods. When the operation of a control system is highly critical due to human safety factors or the high cost of failure in damaged capital or products, the software designers must expend extra effort to validate and verify their software before it can be released. In flight-critical operations, validation and verification are part of the flight certification process.

Many new software technologies exist that can facilitate the development and deployment of complex control systems. For example, component-based architectures [1, 2] promote code reuse, thereby decreasing development and validation time. Moreover, these architectures encourage flexible “plug-and-play” extensibility and evolution of systems. Distributed object computing allows heterogeneous components to interoperate across diverse platforms and network protocols [3, 4]. Open standards for software mean that products from different vendors can interoperate transparently to the user. New advances are being made to enable dynamic reconfiguration and evolution of systems while they are still running [5, 6]. New commu-

Bonnie S. Heck · Linda M. Wills · George J. Vachtsevanos
School of Electrical and Computer Engineering, Georgia Institute of Technology,
Atlanta, GA 30332-0250, USA; e-mail: gJV@ece.gatech.edu

© 2003 IEEE.

K.P. Valavanis (ed.), Applications of Intelligent Control to Engineering Systems, 267–293.
Communication technologies are being developed to allow networked embedded devices to connect to each other and to self-organize [7].

Historically, there has been a disconnect between the control engineers who design the controls and the software engineers who implement them. A thorough understanding of the methodologies used in both disciplines would make the overall process more cohesive. This article gives a tutorial overview of specific new software technologies that are useful for implementing and reusing complex control systems. The focus is on distributed controls, which utilize multiple processors, particularly those that need to be reconfigured either online or offline. This article first discusses technologies for component-based design and reuse of complex control systems, including object-oriented approaches, patterns, and frameworks. It then describes distributed computing and middleware technology for supporting the composition and interaction among control system components that are distributed. The article also presents advances in real-time distributed computing that are needed to support the real-time demands of control systems in coordinating distributed interactions. Finally, it surveys the current technology that is commercially available and on the research horizon for building reusable distributed control systems. The new software technologies discussed in this article can enhance the performance of these existing commercial systems and may form the basis for the next generation of distributed control systems.

11.1 Component-Based Architectures

Consider the engineering term component, which means a part or a subsystem in a complex engineering system. Typically, components are designed to be generic enough to be used in multiple different applications. An engineer might design a system by composing various components, ideally commercial-off-the-shelf (COTS). Similarly, software engineers strive to build software components [1, 2], which are modules that can be used in several applications and can be composed together to build larger applications. Some may be available as COTS. Components that interconnect across a network are called distributed components.

An advantage of software components is that they promote code reuse across many different applications, thereby reducing development and validation time. To construct components with this property, a designer must first assess a range of different applications and then determine generic patterns, which are processing needs and computational solutions to them that are similar across all of the applications [8]. The designer then creates software components implementing the common solutions. Typically, the components can be specialized further in the individual applications.

The software architecture of an entire control system may be component based. As an example, consider the components in a standard hierarchical control system: sensors, low-level control algorithms (such as stabilizing controllers in aircraft or set-point controllers in process control), high-level control algorithms (such as su-