27

# MANAGING FAULT MONITORING AND RECOVERY IN DISTRIBUTED REAL-TIME CONTROL SYSTEMS

Robert W. Brennan and Douglas H. Norrie
*Department of Mechanical and Manufacturing Engineering*
*University of Calgary, 2500 University Dr. N.W. Calgary, T2N 1N4*
*brennan@enme.ucalgary.ca*

*In this paper we describe a holonic approach to managing distributed real-time control applications for manufacturing. The main feature of the architecture are layering (to support functional decomposition) and clustering (to support task propagation).*

## 1. INTRODUCTION

The current trend toward higher levels of automation in the manufacturing and process industries has created a greater reliance on software. In particular, as devices such as controllers, sensors and actuators become "smarter", safety functions that were previously performed by mechanical or electrical interlocks have been assumed by computer software that may reside in a single device or distributed across multiple devices. Failures in these systems can have a significant impact on a company's bottom line (e.g., loss of equipment, production down-time), and more importantly, on the lives of many people.

As a result, as software begins to play an increasingly important role in safety-critical systems, it becomes increasingly important to understand the risks associated with these systems and how these risks can be managed. The objective of the research reported in this paper is to develop software techniques to achieve safe operation of these new distributed computer-based control systems.

We begin with background on distributed real-time control system, then outline the basic requirements for these systems in section 3. Next, we describe our layered architecture to support these basic requirements in section 4 then discuss our current and future work in this area in section 5.

## 2. BACKGROUND

Manufacturing control systems clearly fall in the category of safety-critical systems since these systems should not incur too much risk to persons or equipment. As well,

V. Mařík et al. (eds.), *Knowledge and Technology Integration in Production and Services*
© Springer Science+Business Media New York 2002

the typical requirements of real-time systems such as timeliness, responsiveness, predictability, correctness and robustness are also of fundamental importance.

The primary distinction between non-real-time and real-time systems is that real-time systems tightly link correctness with timeliness. In other words, deadlines must be met under hard real-time (i.e., tasks must finish by a specified time) and soft real-time (i.e., tasks must meet deadlines on average) constraints (Douglass, 1999). Because of the more stringent requirements for latency, reliability and availability (Leveson, 1995), it follows that the step from the non-real-time or soft real-time domain is a large one. This is particularly true in light of the recent trend towards distributed process.

Although there has been a considerable amount of work in the area of fault-tolerant design of industrial control systems (Anderson, 1981; Siewiorek, 1982), fault-tolerant software, and particularly fault-tolerant *distributed* software is a relatively new area of research (Jalote, 1998). Recently, there have been a number of advances in distributed intelligent control that provide the tools to move away from the traditional centralized, scan-based programmable logic controller architecture towards a new architecture for real time distributed intelligent control. In particular, there have been a number of advances recently in programming languages (Lyons, 1998; Wang, 2001), models for distributed control (IEC, 2000) and software methodologies (Lyons, 1998; Odell, 2000). As well, there have been numerous advances in the development of intelligent field devices (e.g., sensors and actuators) that combine built-in processing capabilities with standard communication interfaces. These "Fieldbus" devices have been the focus of the IEC 1158 standard for a number of years, as well as the recent emergence of proprietary solutions such as Siemens Profibus and Allen-Bradley's DeviceNet.

The International Electro-technical Commission (IEC) 61499 standard is one example of this new trend (IEC, 2000). This standard addresses the need for modular software that can be used for distributed industrial process control. In particular, this standard builds on the function block portion of the IEC 61131-3 standard for PLC languages (Lewis, 1996) and extends the function block (FB) language to more adequately meet the requirements of distributed control in a format that is independent of implementation. A detailed description of the IEC 61499 standard is beyond the scope of this paper, however further details concerning this model and its relationship to multi-agent and holonic systems concepts can be found in (Brennan, 2001b).

# 3. REQUIREMENTS FOR DISTRIBUTED CONTROL

In this section we focus on the motivation for our work in distributed real-time control systems, which follows from three key requirements of these systems: (i) control application development, (ii) reconfiguration, and (iii) fault monitoring and recovery.

## 3.1 Basic Requirements

The first and most fundamental requirement is that our system should be capable of allowing the user to develop control applications using IEC 61499 function blocks.

Ideally, the development environment should comply with the standard's "class 2" (IEC, 2000): i.e., the user should be able to create completely new data types, function block types, and configurations of function blocks (Lewis, 2001). As well, to support ease of programming, the system should allow applications to be built from standard libraries of IEC 61499 function blocks as well as specialized, user-defined libraries. Once the control application is developed, the system should then be capable of arranging for compilation of the code into low-level application code and distributing this application code to appropriate resources for execution. For example, "resources" can be physical devices such as a robot, or may be individual microcontrollers on a physical device (e.g., for servo control). Once distributed, these function blocks must be executed, taking account of their timing and precedence relationships. This particular difficult when we consider distributed systems, since correctness must be maintained for applications that run over a number of CPUs, each with its own time base. Finally, if changes are required, the system should be capable of run-time reconfiguration. This may involve simply replacing portions of the running application at the granularity level of an individual function block or, the removal of a function block and the addition of a different function block or group of function blocks.

The configuration and reconfiguration of distributed function block applications is a key area of holonic systems research. This becomes particularly apparent when one considers that one of the fundamental holonic manufacturing systems ideals is the development of systems that can automatically adapt to change. A detailed explanation of our approach to configuration and reconfiguration of these systems, is beyond the scope of this paper, however one may consult (Brennan, 2001a) for further details.

Monitoring and fault recovery is a key requirement of any industrial control system, so it is not surprising that it is also critical for distributed real-time control system design. The purpose of monitoring is to ensure that the control system performs as intended. In other words, this involves ensuring that no unidentified, or latent, faults occur. Fault monitoring is basically the process of watching for failures and errors that may occur when the system is running or that are present (but possibly undetected) in the system itself. Before going any further, we should look at the terms "failures" and "errors" more closely. Failures relate to events that occur at specific times. For example, breakage or excessive wear of mechanical components or overloading of electrical components often manifests failures. Since software does not "break" or "wear" (though, arguably, data may "wear" by becoming obsolete), the notion of "component failures" has little relevance for control software. Errors, however, are a different matter: i.e., an error is an inherent characteristic of the system. As a result, this concept is more relevant to software systems since errors are often manifested in program "bugs".

Systematic errors can be, and almost always are, present in control software. As well, random failures can, and typically do, occur in the controlled system. Given these eventualities, the control system must be capable of recovering from the resulting faults. As a result, the types of responsibilities that our control system will have are: (i) diagnosis of program execution, (ii) monitoring for exceptions that are thrown by function block code during execution, and (iii) monitoring the system state for inconsistencies (e.g., deadline control).

To achieve a safe system, typically two general concepts are used (Leveson, 1995). The first approach involves separating the fault monitoring and recovery code (i.e., the "safety channels") from the control application code. This decomposition technique is typically referred to as the "firewall concept" (Leveson, 1995), since one may think of the safety channels as providing a firewall for the control code. The second approach involves applying redundancy in the control system. This is a very common means of fault recovery and can be achieved in two basic ways: homogeneous redundancy and diverse redundancy. Homogeneous redundancy, is the most basic form of redundancy, and involves "clones" or exact replicas of code to be used as backup code in the case of a failure. The main disadvantage with this technique is that it only protects against random failures. For example if a piece of hardware fails, a copy of the code that was running on the failed device can be moved to another device to allow the system to continue operating. However, if the failure was caused by a systematic error in the software (e.g., a command that causes the process to go into a deadlock state), this technique will not work: i.e., the same failure will occur on the new device. To overcome this problem and provide protection against random and systematic failures, diverse redundancy can be used. With this approach, the redundant code is implemented in a different way. For example, the redundant code for a PID controller could be implemented using fuzzy logic.

In the next sub-section, we propose an approach that builds on the "firewall concept" in order to address the issue of system safety (i.e., fault monitoring and recovery) as well as the basic requirements of control application development and reconfiguration for distributed real-time systems.

### 3.2 Methodology

The research presented in this paper is intended to address the basic requirements noted above by taking advantage of recent advances distributed control system models, software and hardware to realize a distributed process control system with intelligent control components to work in safety-critical environments.

In remainder of the paper, we describe a layered architecture to manage fault monitoring and recovery in distributed control systems based on the notion of "holonic agents" (Marik, 2001). Through their work with the IEC 61499 function block model, members of the Holonic Manufacturing Systems Consortium (HMS, 2002) came to the realization that the best approach is to encapsulate the function block solution into higher-level software when and where it is required to enable more sophisticated reasoning and a richer knowledge representation than function blocks alone. These resulting holonic agents can then integrate a holonic part (for hard real-time) and software agent part (responsible for higher level, soft real-time or non-real-time intelligent decision-making).

This paper extends this idea to a multi-layer architecture consisting of four temporally decomposed layers of agents and devices illustrated in Figure 1: execution control (EC), control execution (EC), execution (E), and hardware. As one moves down the layers, time scales become shorter and real-time constraints change from soft to hard real-time; as well, the degree of agency decreases (i.e., higher-level agents are more sophisticated but slower, while lower-level agents are fast and light-weight).

The EC and CE layers shown in Figure 1 support two fundamental system requirements: control application management, and fault detection and recovery. Since the real-time holonic control system is intended to meet hard real-time requirements, a temporal decomposition of the agents used for this purpose is necessary.

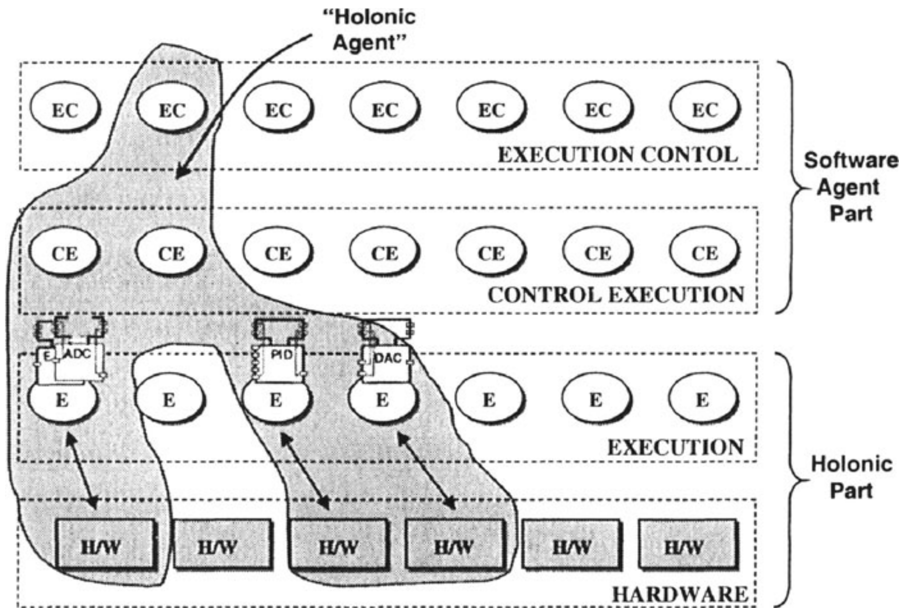


Figure 1 – Layered Architecture

The EC layer is composed of agents, or "holonic units" (i.e., the ovals shown in Figure 1), that are responsible for arranging control applications for execution as well as planning for reconfiguration, fault monitoring and fault detection. In general, CE holonic units are responsible for controlling what is being executed. In other words, the CE holonic units are concerned with distributing execution control code to the appropriate resources (shown as "H/W" in Figure 1), performing basic monitoring and alerts, and handling low-level fault recovery procedures. In cases where more sophisticated fault recovery decision-making is required, the CE layer will consult with holonic units in the EC layer (who may even have to consult higher-level agents). In the next section, we describe this decomposition in more detail.

# 4. AN ARCHITECTURE FOR DISTRIBUTED REAL-TIME CONTROL

One of the main features of our architecture for distributed real-time control is the concept of layering. The idea here is to identify various autonomous layers that appear to interact through API's (application program interfaces). On each layer, code is run asynchronously, resulting in clear functional separation across the layers. As will be discussed in section 4.1, clustering is another important feature of the architecture. In other words, everything happens through task propagation (and concurrently) through cluster formation.

Two general approaches can be used to execute applications using this general approach. First, all function blocks below the application level can be handled by a single level of controllers. Alternatively, function blocks can be distributed over the system for execution across a web of "holonic controllers" (i.e., the shaded area over the Execution and Hardware layers in Figure 1). In this case, higher-level function blocks (and/or software agents) run on "application level" controllers, middle-level function blocks run on "execution level" controllers, and low-level function blocks run on "control execution level" controllers and base level ("execution") controllers.

For the remainder of this section we can think of each layer as an independent set of controllers. Communication between the layers is facilitated by the layer interfaces.

## 4.1 Task Propagation

In Figure 1, the shaded boundary that extends from the EC layer down to the hardware layer is an example of a control application that is being run on a cluster of devices (for e.g., CNC's, AGV's, or Robots). The devices represent a virtual cluster (or virtual cell) of devices with a control application that is distributed across specific resources in each of the devices.

Figure 2 illustrates the concepts of "layering" and "clustering". First, a task is initiated by a command from a higher-level. For example, at the planning and scheduling level (not shown in Figure 2), a request may be made to move a part from point A to point B. At this stage, the first cluster of EC/CE agents is formed to handle the task execution. Next, a second cluster is formed, which represents the distribution of the control application. Finally, each bit of distributed code is executed on a specific hardware platform as shown by cluster 3.

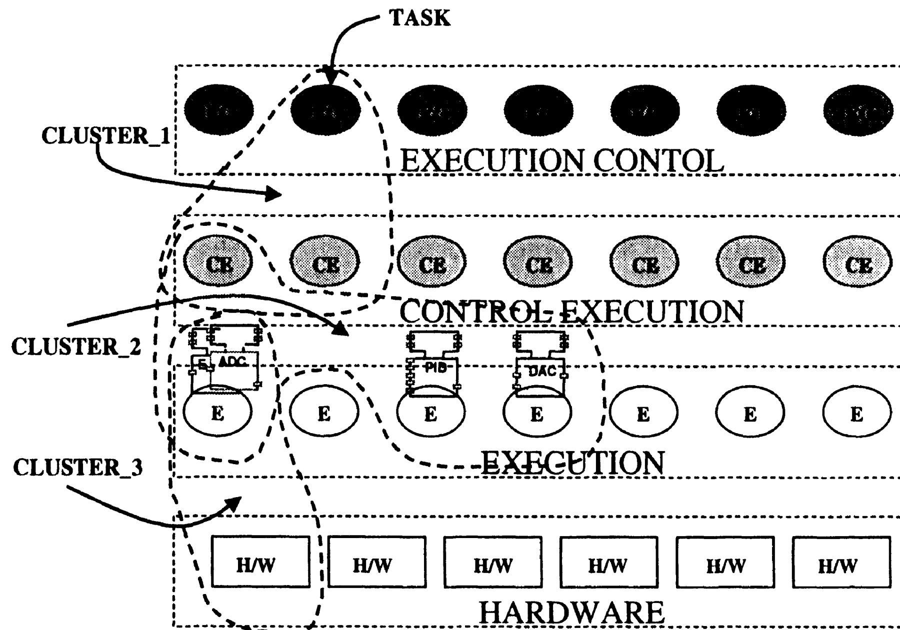


Figure 2 – Task Propagation

From a top-down perspective, layering and clustering allow us to manage the functional decomposition of high-level tasks and automatically determining the required distribution of function blocks. As will be discussed in the next sub-section, the resulting clusters can also be used for bottom-up fault-recovery management.

### 4.2 Fault Monitoring and Recovery

A primary advantage of this approach is that we can use the task sub-clusters illustrated in Figure 2 to track back as far as is required for fault monitoring and recovery. For example, Figure 3 illustrates what happens when a fault occurs at the hardware level. In this case, the fault is first caught at the execution level. For example, data corruption checks or reasonableness checks (i.e., comparisons using redundant software) may be used to catch the fault. The fault is then reported to the next higher level.

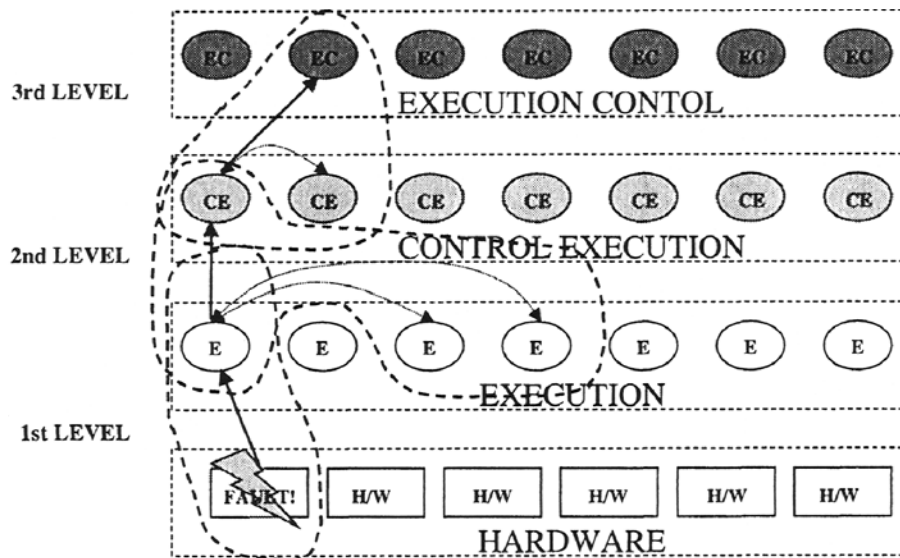


Figure 3 – Fault Monitoring and Recovery

Next, the CE level receives the fault information where the CE agents at this level first attempt more sophisticated means of recovery. For example, feedback error detection or feed-forward error detection may be used. In the first case, the fault is identified, but is not corrected. Instead, the sub-task (at the execution level) is redone or the hardware is placed in a fail-safe state. In the second case, the CE agents try to correct the error and continue processing. For example, "recovery" at this level may involve reconstruction of corrupt data. If the fault is unrecoverable however, the hardware would be placed in a fail-safe state and the CE agents would report to the execution control level.

Finally, at the EC level, more sophisticated techniques such as homogeneous or diverse redundancy may be used to recover from the fault. Of course, if this level is incapable of recovering (e.g., an operation cannot be done, so the batch will have to be rescheduled), higher levels may be consulted.

## 5. CONCLUSIONS

In this paper, we have provided a general overview of a layered architecture to manage the key requirements of distributed real-time control: i.e., (i) control application development, (ii) reconfiguration, and (iii) fault monitoring and recovery. At the most basic level, the architecture builds on the basic distinction between agents and holons: i.e., that a holon consist of a software part and a physical part. When considering the competing requirements of control application execution and system fault monitoring and recovery however, the architecture builds on the safety-critical systems' firewall concept. In other words, our layered architecture allows control channels (managed by a top-down task decomposition process) to be separated from safety channels (managed by a bottom-up fault monitoring and recovery process). Currently, we are working on a detailed approach to achieve dynamic and automatic reconfiguration of distributed real-time control systems (Zhang, 2001). Interestingly, we have found that the basic concepts from the general model reported in this paper are equally applicable to lower-level models. For example, in order to manage function block application reconfiguration, we have found that it is useful to also separate execution channels from "safety" channels. As a result, we represent function blocks using two flow paths: (i) an "execution control path" for control application management, and (ii) a "configuration control path" for configuration and reconfiguration control.

## 4. REFERENCES


1. Anderson, T, Lee, PA. Fault Tolerance Principles and Practice, Prentice Hall, 1981.
2. Brennan, RW, Fletcher, M, Norrie, DH. Reconfiguring real-time holonic manufacturing systems. Twelfth International Workshop on Database and Expert Systems Applications, IEEE Computer Society, pp. 611-615, 2001a.
3. Brennan, RW, Norrie, DH. Agents, holons and function blocks: distributed intelligent control in manufacturing. Journal of Applied Systems Studies Special Issue on Industrial Applications of Multi-Agent and Holonic Systems 2001b; 2(1): 1-19.
4. Douglass, B. Doing Hard Time: Developing Real-time Systems with UML, Objects, Frameworks, and Patterns, Addison-Wesley, 1999.
5. Holonic Manufacturing Systems Consortium, Website, http://hms.ifw.uni-hannover.de/, 2002.
6. IEC TC65/WG6, Voting Draft: Function Blocks for Industrial Process-Measurement and Control Systems, Part 1 Architecture, International Electrotechnical Commission, 2000.
7. Jalote, P. Fault Tolerance in Distributed Systems, Prentice Hall, 1998.
8. Leveson, N. Safeware, Addison-Wesley, 1995.
9. Lewis, R.Modelling Control Systems using IEC 61499: Applying Function Blocks to Distributed Systems, IEE, 2001.
10. Lewis, R. Programming Industrial Control Systems using IEC 1131-3, IEE, 1996.
11. Lyons, A. UML for real-time overview. Technical Report of ObjecTime Ltd, 1998.
12. Marik, V, Pechoucek, M. Holons and agents: recent developments and mutual impacts. Twelfth International Workshop on Database and Expert Systems Applications, IEEE Computer Society, 605-607, 2001.
13. Odell, J, Parunak, H, Bauer, B. Extending UML for agents. ERIM Center for Electronic Commerce, http://www.erim.org/~vparunak, 2000.
14. Siewiorek, DP, Wsarz, R. The Theory and Practice of Reliable System Design, Digital Press, 1982.
15. Wang, L, Brennan, RW, Balasubramanian, S, Norrie, DH. Realizing holonic control with function blocks. Integrated Computer-Aided Engineering 2001; 8(1): 81-93.
16. Zhang, X, Brennan, RW, Xu, Y, Norrie, DH. Runtime adaptability of a concurrent function block model for a real-time holonic controller. IEEE Systems, Man, and Cybernetics 2001. 164-168.