5

CONTINGENCIES-BASED RECONFIGURATION OF HOLONIC CONTROL DEVICES

Scott Olsen, Jason J. Scarlett, Robert W. Brennan, and Douglas H. Norrie
Department of Mechanical and Manufacturing Engineering
University of Calgary, 2500 University Dr. N.W. Calgary, CANADA T2N 1N4

In this paper, we propose a dynamic approach to programmable logic controller (PLC) reconfiguration that is based on the IEC 61499 standard for distributed, real-time control systems. With this form of reconfiguration control, contingencies are made for all possible changes that may occur. We illustrate this approach with a simple system configuration that uses Rockwell Automation's Function Block Development Kit (FBDK) for the software implementation and Dallas Semiconductor's Tiny InterNet Interface (TINI) for the hardware implementation.

1. INTRODUCTION

By definition, "holons" contain both an information processing part and a physical part (Gruver et al., 2001). Moreover, "holonic systems" are essentially adaptive agent-machine systems. As a result, it is not surprising that the Holonic Manufacturing Systems Consortium (2004) has a work group devoted to these software/hardware devices or Holonic Control Devices (HCD). This paper focuses on the "adaptive" aspect of these agent-machine systems. In particular, we investigate how emerging software and hardware technologies can be taken advantage of to create systems at the device level that can dynamically reconfigure themselves in response to changes in the manufacturing environment (e.g., device malfunctions or the addition and/or removal of equipment).

Reconfiguration of conventional industrial controllers such as PLCs (programmable logic controllers) involves a process of first editing the control software offline while the system is running, then committing the change to the running control program. When the change is committed, severe disruptions and instability can occur as a result of high coupling between elements of the control software and inconsistent real-time synchronization. For example, a change to an output statement can cause a chain of unanticipated events to occur throughout a ladder logic program as a result of high coupling between various rungs in the program; a change to a PID (proportional/integral/derivative) function block can result in instability when process or control values are not properly synchronized.

In this paper, we propose a dynamic approach to PLC reconfiguration that is based on the IEC 61499 standard (IEC, 2000) for distributed, real-time control systems. With this form of reconfiguration control, contingencies are made for all possible changes that may occur. In other words, alternate configurations are pre-programmed based on the system designer's understanding of the current configuration, possible faults that may occur, and possible means of recovery. This approach uses pre-defined reconfiguration tables that, in the event of a device failure, allow the affected portions of an application to be moved to different devices by selecting an appropriate reconfiguration table.

The paper begins with an overview of our contingencies-based reconfiguration model. In order to implement this approach, we develop an IEC 61499 based reconfiguration management service that allows function block applications to be reconfigured dynamically (i.e., the management services ensure that the HCD is properly synchronised during reconfiguration). This reconfiguration manager also serves as an interface to higher-level software to enable intelligent reconfiguration (e.g., the use of multiagent techniques to allow the system to reconfigure automatically in response to change) (Brennan and Norrie, 2002).

Next, we illustrate this approach with a simple system configuration that uses Rockwell Automation's Function Block Development Kit (FBDK) (Christensen, 2004) for the software implementation and Dallas Semiconductor's Tiny InterNet Interface (TINI) (Loomis, 2001) for the hardware implementation. The paper concludes with a discussion of on how the reconfiguration process can be managed in a resource-constrained environment (i.e., such as on the TINI board), the limitations of Java for real-time distributed control, and the possibilities for intelligent reconfiguration in this type of system.

2. DESIGNING FOR RECONFIGURATION

2.1 A Contingencies Approaches to Reconfiguration

With this form of reconfiguration control, contingencies are made for all possible changes that may occur. In other words, alternate configurations are pre-programmed based on the system designer's understanding of the current configuration, possible faults that may occur, and possible means of recovery.

This approach uses a library of pre-defined configurations as shown in Figure 1. For example, in the event of a device failure, the affected portions of an application could be moved to different devices by selecting an appropriate configuration. As well, this detailed representation of the function block interconnections would allow higher-level agents to access the information required to make a smooth transition from one configuration to another, thus enabling dynamic reconfiguration.

2.2 An IEC 61499 Based Reconfiguration Management Service

An IEC 61499 based "reconfiguration manager" was developed to address the need for an interface between upper level agents (e.g., scheduling, fault monitoring, configuration) and lower control applications for dynamic reconfiguration of

distributed systems. With respect to the manufacturing control system as a whole, external agents responsible for failure monitoring and system wide reconfiguration interact with each other as well as with the reconfiguration manager and the system to be controlled.

The reconfiguration manager must have certain capabilities in order to effectively link the higher-level agents to the lower level machine control. The reconfiguration manager running on the controller must be able to receive messages from agents in order to apply the appropriate control application to the controlled process, as shown in Figure 1. For example, scheduling agents (SA) in charge of scheduling parts to be processed and machine agents responsible for monitoring the status of the processing equipment may send information to a configuration agent (CA) that makes the decision of which control application should be implemented on the controller. Through the I/O capabilities of the controller, the control application may control and/or monitor a process that is accomplishing a specific task.

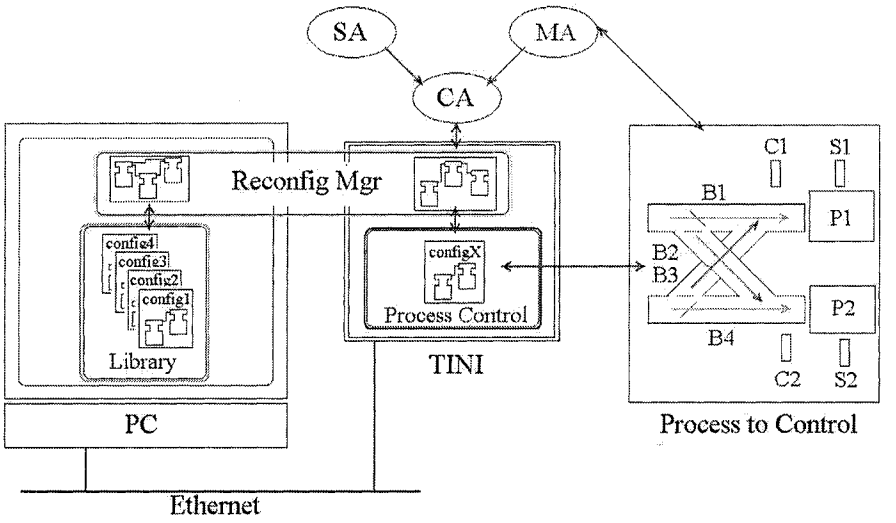

Figure 1 – The reconfiguration manager implementation

The basic function of the reconfiguration manager is to load and unload new control applications on the controller hardware at the request of outside agents. During the transition period when a control application is unloaded and a new one is being loaded, the reconfiguration manager must have the ability to maintain a transitional state for the controlled process to remain in. It should be noted that the control applications must have the built-in ability to save the state information of the controlled process, and also retrieve that state information for the next control application if necessary. These features allow for continuous smooth and stable performance of the controlled process.

Loading a new control application requires access to the library where all of the contingent control applications are stored. Since the control application only defines how function blocks are connected together and the values of any external variables,

there must be access to the library where the class files are stored. These libraries may each be located on different hardware platforms. In order to be effective, the reconfiguration manager must perform its function under real-time constraints.

The reconfiguration manager is a distributed application running concurrently on the embedded control platform (i.e., the TINI platform) and the PC as shown in Figure 1. Communication between the PC and TINI is through an Ethernet connection. In the next section, we provide further details on the prototyping environment as well as some basic experiments that were conducted to test this approach to reconfiguration control.

3. EXPERIMENTS

3.1 The Prototyping Environment

The prototyping environment that is currently being used for our experiments with IEC 61499 consists of function block application software and a Java-based control platform.

The software component of our prototyping environment, Function Block Development Kit (FBDK) (Christensen, 2004), was developed by Rockwell Automation primarily for testing IEC 61499 concepts in a simulated environment. For our experiments, a MANAGER function block is used to provide the IEC 61499 device management services on our controller platforms. For example, this function block provides services to allow function block instances to be created, deleted, started, killed, etc. and also provides information on function block status (e.g., READY, INVALID_STATE, OVERFLOW, etc.). This allows distributed applications to be developed using FBDK and run (and managed) on a network of controllers (rather than just simulated on a single PC).

The hardware component of our prototyping environment consists of a Dallas Semiconductors Tiny InterNet Interface (TINI) (Loomis, 2001) board running on a Taylec TutorIO prototyping board. The TINI board includes a DS80C390 microcontroller (an Intel 8051 derivative) that supports JDK 1.2 (Java Development Kit) applications and also supports several forms of I/O such as discrete and analogue I/O, serial, Ethernet, 1-Wire and Controller Area Network (CAN). In order to experiment with the TINI board, the TutorIO board provides interfaces to the I/O (e.g., a two-line LCD, LEDs, etc.).

3.2 The Test Scenarios

The experimental set-up is composed of a conveyor belt system for transporting parts to a manufacturing process. As parts arrive at each process, they are counted so that when a certain number of them have arrived, the conveyor is stopped and the process is applied to a batch of parts. When the process has been completed, a signal is sent to start the conveyor again.

This particular conceptual manufacturing system was chosen so that it would be flexible enough to allow for various configurations but not overly complex. A less complex system allows for the focus to be placed on demonstrating the key concepts associated with the reconfiguration process rather than on the control application

itself. Also, due to the limitations of the TINI, particularly with its relatively slow application loading times, a less complicated control application is desirable.

To compare the results of the reconfiguration manager and control applications running the TINI, a similar system was implemented entirely on the PC. A virtual device was set up on the PC with FBDK to simulate the TINI in order to run a modified version of the reconfiguration manager and the control applications. The inputs and outputs from the Taylec board were simulated with virtual lights and buttons that appear on the PC screen. The simulated versions of the reconfiguration manager and the control applications running on the "virtual TINI" have identical functionality as the TINI-based applications. This allows a reasonable comparison of the two systems and a meaningful assessment of the prototyping environment.

This part of the experiments was motivated by the limitations of the TINI platform. For example, one of the limitations is a result of a limit on the number of Publisher/Subscriber pairs a TINI can support. This is a result of the number of threads that a TINI can support. Since PCs typically have much higher processing capabilities in comparison to embedded platforms, the PC comparison provides a lower bound on these latencies. In terms of real-time performance, the PC may not possess superior capabilities. However, in this case, the limitations relate more to processor resource limitations which are rapidly being overcome by new platforms (discussed in section 4).

The three test scenarios reported in this paper are illustrated in Figure 2. The initial configuration, illustrated in Figure 2(a), involves a single conveyor (B1) that transports parts to a holding area where a process (P1) is applied to a batch of parts. The parts are counted by sensor C1 as they pass along the conveyor belt. When the process has been completed, sensor S1 sends a signal indicating that the P1 is ready for a new batch of parts.

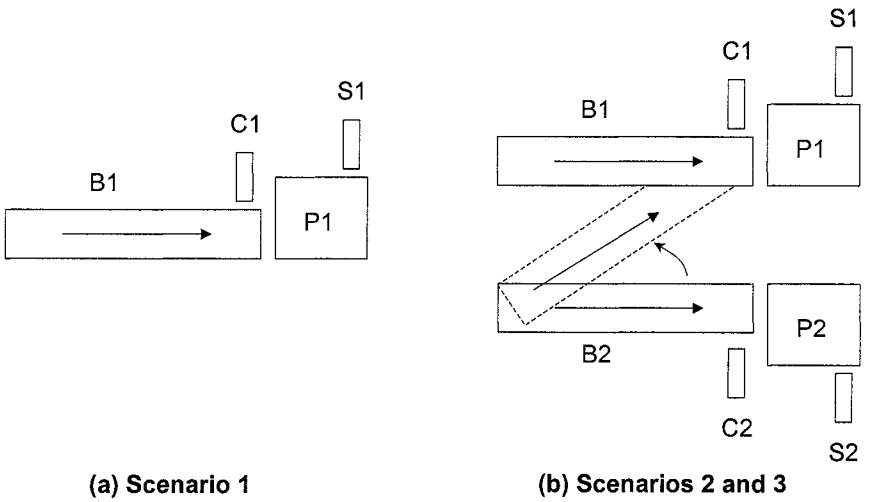

Figure 2 – The test scenarios

The second scenario, shown in Figure 2(b), depicts the situation where a second process (P2) is added. This contingency may be required in the event of additional equipment is introduced to the system or in the event that the controller for the second process has failed.

Finally, the third scenario, also illustrated in Figure 2(b), involves the case where a failure at P2, or in the event of rescheduling by a higher-level agent (e.g., to balance the two lines shown in Figure 4(b)). In this case, the conveyor feeding P2 is moved to feed P1, or a third conveyor is added to divert parts to P1 (shown by the dashed conveyor in Figure 2(b).

3.3 Experimental Results

In order to quantify the real-time performance of the reconfiguration manager, the time to load and run different applications was measured. Loading times were measured for applications running on the TINI and for the simulated 'virtual TINI' system to allow for a reasonable assessment of the performance on the prototyping platform. Two different TINI boards were used to perform the reconfiguration experiments, with nearly identical results from each one.

In Figure 3 we show the experimental results for the TINI platform. For these results, the application launch time represents the time to kill the application running on the TINI platform, FTP the contingent control application to the TINI platform, and then launch this new application. In order to compare the three scenarios described in the previous section, we calculate an application complexity metric that is based on application size (in KB), number of function block connections, and number of function blocks. This metric is normalized with respect to the initial scenario (i.e., this scenario has a complexity of 1). For example, scenario 1 has an application size s_1 = 5.02 KB and it uses c_1 = 27 connections to connect b_1 = 11 function blocks. As a result, scenario 3 (s_3 = 7.91 KB, c_3 = 44, b_3 = 20) has a relative complexity of 1.67: i.e.,

$$\text{Complexity of scenario 3} = (s_3 / s_1 + c_3 / c_1 + b_3 / b_1)/3 = 1.67$$

Figure 4 shows the results for the 'virtual TINI' system noted previously. For these experiments, all of the same steps noted above were performed, however the control application was run on a 400 MHz, Pentium II processor.

Comparing the launch times to the relative complexity of the control applications in Figures 3 and 4, it is apparent that there is a correlation between launch time and application size. The launch time increases significantly with the increase in application size. Unfortunately, the launch times on the TINI platform are too slow for practical applications. This is partially a result of the increased processing requirements placed on the TINI platform by the reconfiguration manager process (shown in Figure 1). However, the overall performance of is still quite slow without this process running. For example, scenario 1 takes 85 seconds to launch and scenario 3 takes 102 seconds to launch on the TINI without the reconfiguration manager running.

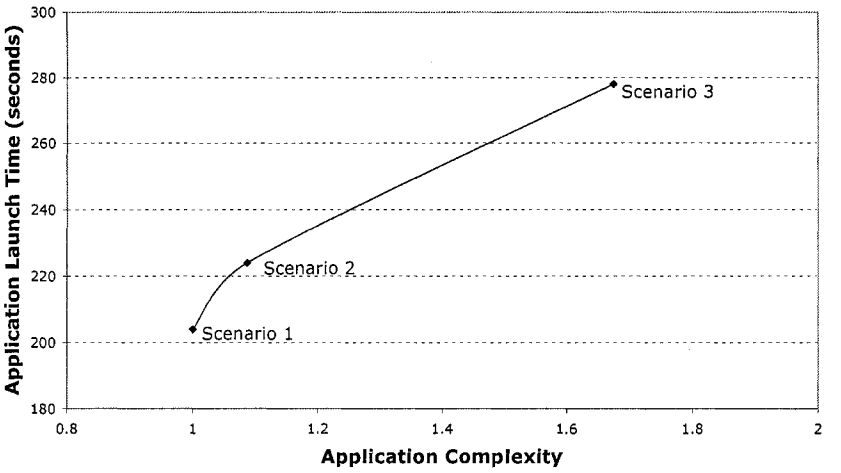

Figure 3 – Application launch time for the TINI platform

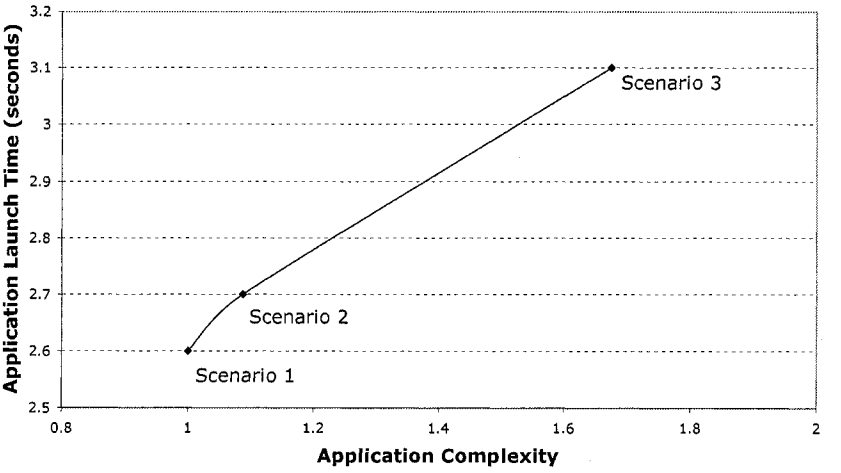

Figure 4 – Application launch time for the PC platform

4. CONCLUSIONS

It appears from the results reported in the previous section that the main limitation of this reconfiguration approach lies in the application launching process as illustrated in Figure 3. In particular, when reconfiguration is required, the existing control application is killed, a new application is loaded, and finally the new application is launched. Once the control application is launched, the TINI

platform's real-time performance is quite acceptable (please refer to Rumpl et al., (2001) for more information on the TINI's run-time performance).

Reconfiguration should not necessarily require a complete replacement of the existing application however. For example, the changes required to move from a control application for scenario 1 to one for scenario 2 or 3 should only require the addition of a small number of function blocks and/or the "re-wiring" of variable and event connections. Unfortunately, the TINI Java Virtual Machine (JVM) supports a limited implementation of JDK 1.2, and as a result, does not support this form of reconfiguration. In particular, the TINI JVM does not support dynamic class loading and serialization.

In order to address this implementation issue, the authors are currently investigating alternative embedded Java-based platforms such as the Systronix aJile Euroboard (or SaJe) (SaJe, 2004) and the ImSys Simple Network Application Platform (or SNAP) (ImSys, 2004). Our work in this area is focusing on both dynamic and intelligent reconfiguration issues. For example, given the increased memory and processing speed of these platforms, we will be able to more closely investigate the timing issues associated with reconfiguration.

As well, the "re-wiring" approach noted above will be further investigated. The IEC 61499 standard's support of XML descriptions is particularly promising in this area (IEC, 2000). This format allows unambiguous specifications to be written for function block applications that can be used for initial system configuration as well as subsequent reconfiguration. In other words, changes in an application's configuration can be specified using a well-formed XML document; the reconfiguration manager's job is then to parse the XML description and to make the necessary changes to the application (e.g., changing connections, adding/removing function blocks, etc.). The anticipated advantages of this approach are that it should overcome the application loading issues described in this paper and also open the door for more intelligent approaches to reconfiguration (e.g., using higher level configuration agents to reason about new configurations rather than relying on pre-defined contingencies).

5. REFERENCES

1. Brennan, R.W. and Norrie, D.H. "Managing fault monitoring and recovery in distributed real-time control systems," *5th IEEE/IFIP International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services*, Cancun, Mexico, pp. 247-254, 2002.
2. Christensen, J.H. *Function Block Development Kit*, holobloc.com, 2004.
3. Gruver, W., Kotak, D., van Leeuwen, E., Norrie, D. (2001) "Holonic manufacturing systems – phase 2", *the International IMS Project Forum 2001*, Ascona, Switzerland.
4. HMS, *Holonic Manufacturing Systems Consortium Web Site*, http://hms.ifw.uni-hannover.de/, 2004.
5. IEC TC65/WG6 (2000) Voting Draft – Publicly Available Specification - Function Blocks for Industrial Process-measurement and Control Systems, Part 1-Architecture, International Electrotechnical Commission.
6. ImSys, *Web Site*, http://www.imsys.se, 2004.
7. Loomis, D. *The TINI Specification and Developer's Guide*, Pearson, 2001.
8. Systronix, *Web Site*, http://www.systronix.com, 2004.