A Pattern-based Approach to Distributed Control System Design

Robert W. Brennan, James Wang, and Douglas H. Norrie
Department of Mechanical and Manufacturing Engineering
University of Calgary, 2500 University Dr. N.W., Calgary, AB, T2N 1N4
Phone: +1-403-220-5798
Email: rbrennan@ucalgary.ca

Abstract

This paper focuses on the use of software design patterns to assist in the design and development of distributed control software. In particular, the Model-View-Controller (MVC) framework is used to develop IEC 61499 function block applications. In order to test this approach, a simple example of a distributed control application is implemented on a Java-based embedded control platform.

1. Introduction

The use of patterns for design has a long history, with origins in early architectural design where patterns were used to systematically express key architectural design elements [1]. From an engineering perspective, a design pattern is a specific solution to a design problem. Rather than “reinventing the wheel” each time a new design is performed, engineers will typically re-use well-known solutions to common engineering problems (i.e., “design patterns”) as part of the overall design solution. For example, in the control engineering domain, common arrangements of plant, controller, and feedback loop are used for various classes of control problems.

Although the notion of design patterns is not new to the general design community, it is a relatively new and emerging practice in software engineering. For example, in the late 1980’s papers started to appear in object-oriented programming conferences on the classification of well-tested solutions to common software engineering problems (i.e., “software design patterns”). In 1995 Gamma et al. [2] formalized this approach in their book Design Patterns: Elements of Reusable Object-oriented Software. In recent years, this work has generated a considerable amount of interest in the general software engineering community, however very little work has been done on using software design patterns to solve computer-based control problems.

In this paper we investigate the use of software design patterns for control engineering problems. In particular we focus on the question, how can distributed agent-based techniques be applied at the real-time control level? One approach that looks particularly promising for developing these systems is the Model-View-Controller (MVC) user interface framework [3] that focuses on the development, simulation and deployment of object-oriented systems. In this paper, we apply the MVC framework to a simple manufacturing control problem in order to develop a real-time distributed controller based on an object-oriented control language. In this case we implement our solution with IEC 61499 function blocks [4], however this approach could also be extended to other function block-based control languages such as IEC 61131-3 [5] and Simulink [6].

We begin with an overview of real-time distributed control systems and models, as well as the MVC framework. Next, we introduce our test system then describe the development of our distributed controller. Finally, we give a summary of our experience with our prototype system as well as our thoughts on the use of the MVC framework in this domain.

2. Real-time distributed control systems

Although manufacturing technology has become increasingly sophisticated to deal with the competing objectives of cost, quality and time (e.g., through advanced robotics and computer numerical control), without adequate control, the result is often a collection of “islands of automation” that lack the necessary integration for truly responsive behaviour. As a result, new control software and hardware approaches are required to realize a system that is flexible (i.e., capable of reconfiguration) and
responsive (i.e., capable of recovering from disturbances).

Recently, there have been a number of advances in this area that provide the tools to move away from the traditional centralized, scan-based programmable logic controller (PLC) architecture towards a new architecture for real time distributed intelligent control. In particular, there have been a number of advances recently in programming languages [5], models for distributed control [4] and software methodologies [7].

In this section, we begin by looking at the IEC 61499 model for distributed intelligent control, which serves as the basis for our distributed controller design. We then provide a brief overview of the MVC framework.

2.1. The IEC 61499 function block model

The International Electro-technical Commission (IEC) 61499 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 [5] and extends the function block (FB) language to more adequately meet the requirements of distributed control in a format that is independent of implementation.

The IEC developed the Function Block Architecture as a new standard to model industrial process control systems using decentralization and hard real-time design philosophies. The architecture permits access to the controlled manufacturing process via an IEC 61499 system that contains an organization of devices. The five main models of the IEC 61499 standard are organized in increasing levels of granularity: system model, device model, resource model, application model, and function block model.

Each device comprises of one or more resources (i.e., processor with memory) and one or more interfaces. Interfaces enable the device to interact with either the controlled manufacturing process (via a process interface) or with other devices through a communication interface. Resources are logical entities with independent control over their operations including the scheduling of their tasks. A resource can be created, configured etc (as part of the system’s life-cycle) via a management model.

Applications (software functional units spanning one or more resources and over one or more devices) are networks of function blocks (FB) and variables connected by data and event flows. Such applications aid the modeling of cooperation between the autonomous agents.

Finally, the IEC 61499 function block model is shown in Figure 1. Function blocks receive events/data from interfaces, process them by executing algorithms and produce outputs, all handled by an event control chart. In developing control applications with IEC 61499, the function block can be thought of in terms of an “enhanced” object. Like recent object-oriented and agent-based models for manufacturing system control, the IEC 61499 function block shares many of the characteristics of the traditional objects and agents used to develop these applications (e.g., a traditional object focuses on data abstraction, encapsulation, modularity, and inheritance).

The function block is enhanced through its recognition of two very specific kinds of messages: data messages (which one would expect of a traditional object) and event messages (which are used to schedule the execution of an algorithm). The resulting focus on process abstraction and synchronization makes this approach particularly suitable for control of an environment that is concurrent, asynchronous and distributed.

![Figure 1. The IEC 61499 function block model [4]](image-url)

IEC 61499 primarily focuses on models and definitions that allow distributed applications to be developed using function blocks. To actually create and manage these systems however, further guidance is required. In particular, the standard does not provide any guidance concerning the actual implementation of function block applications. For example, low-level communication protocol requirements such as how to load, initiate and configure function blocks are not addressed. As well, the standard does not address how function block applications are compiled, downloaded, and stored.
In the next section, we provide an example of how these implementation issues can be addressed. First however, we look at the more general issue of developing function block applications.

2.2. The Model-View-Controller framework

The Model-View-Controller (MVC) framework is arguably the most well-known approach to designing interactive software systems using object-oriented or agent-based technology [3]. The basic idea is to divide an interactive application into three components: (1) a Model that contains the core functionality of the application and the data, (2) a View that is used to display information to users, and (3) a Controller that is used to handle user input.

Although the MVC framework was developed primarily to assist with the development of application that must interact with users through a graphical interface, there has recently been interest in using this technique to assist with the development of distributed software systems that must interact with physical devices: i.e., control systems. For example, [8] has proposed an extension of the MVC framework to assist with the development of IEC 61499 function block applications. In his extended framework he suggests that each of the MVC elements would be represented by an instance of a function block type: i.e., the Model function block encapsulates the time dependent and logical behavior of the system or device being controlled, the View function block encapsulates the graphical display associated with one or more Model types, and the Controller function block encapsulates the control functions to be performed on one or more instances of Model types.

In the next section, we investigate how this approach can be used to implement a distributed control application for a simple manufacturing system.

3. Distributed control system development

3.1. The prototype

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) [9], 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 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, shown in Figure 2, consists of a Dallas Semiconductors Tiny InterNet Interface (TINI) board [10] (Figure 2(a)) running on a Taylec TutorIO prototyping board (Figure 2(b)). The TINI board includes a DS80C390 microcontroller 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.).

![Figure 2. The prototyping hardware](image-url)
outputs are LED’s however, the TutorIO board allows one to bypass these on-board inputs/outputs and use external inputs/outputs (this approach was used for this paper).

3.2. Problem description

The experimental manufacturing system investigated in this paper consists of a conveyor belt unit and a vertical dispensing unit as illustrated in Figure 4. The process initiates as raw components are fed onto the left side of the conveyor one at the time. Raw components then stop under a dispensing station where a vertical dispenser attaches a part to create an assembled component. As the dispensing unit returns back to its initial position, the sensor which resides on the top dispensing unit signals the controller to reactivate the movement of the conveyor belt until the assembled component move to the roll-off roller. The whole process is repeated when the next recipient component arrives onto the left-hand side of the conveyor belt.

3.3. Using the MVC framework

Rather than developing the control application immediately as is the common practice in most industrial control applications, we follow the Model-View-Controller framework, which requires first developing a View (a graphical display of the system)
then a Model (time-dependent logical behavior of the system) of the experimental manufacturing system. We can then use this simulation model (i.e., the combined Model and View modules) to develop our Controller and HMI.

Using this approach, we start with a number of views of the process being controlled. In this case, the system View shown in Figure 5 provides a graphical display of the conveyor system and the dispenser channel.

For consistency, we chose to develop all of the software using IEC 61499 function block applications however, the View and Model aspects of the framework could be developed using any general purpose programming language (e.g., Java, C++).

Next, for each view we develop and test one or more Models capable of simulating the time-dependent logical behavior of the experimental system. Function block applications were developed to simulate the vertical movement of the dispenser and the part, and the horizontal movement of the conveyor and the component as illustrated in Figure 6.

An example of the function block code for the Model of the dispenser is shown in Figure 7. The main functionality of the SOL_MDLL function block shown in this figure is to allow an image of a shaft-like object to be shown dynamically in the previously defined View display. As well, SOL_MDLL simulates sensor functionalities, which enable messages to be sent when the shaft moves either to the home position or the bottom-end position (the sensors are illustrated by the small circles on the channels in Figure 5).

The CNV_MDLL function block models the part for the dispenser process. Sensing functionality has also been added to the CNV_MDLL function block: it allows a message to be sent when an object is being transported to the end of the channel section.

Once we have verified that the model works correctly, we can develop our controller using the simulation of the physical system as illustrated in Figure 8.
Figure 8. Controller development

For this work, we used the Real-time Object-oriented Modeling (ROOM) methodology [7] to assist us with controller development. For example, the ROOM state diagram in the upper right of Figure 8 specifies the controller behavior. In this case, the ROOM state diagram serves as a guide for the state behavior of the dispenser controller module. However, the behavior is implemented by the specific controller software that is chosen for the project (e.g., IEC 61499 function blocks, IEC 61131-3 ladder diagrams, etc.).

When we are satisfied with the behavior of our controller, we can then replace the Model and View with the physical device as is shown in Figure 9. As can be seen in this figure, the physical system implemented using a Lego™ model.

4. Discussion

The test results are very promising, indicating that the MVC framework, which effectively requires a simulation model of the system (Model and View) to be developed before the physical implementation, eases the process of distributed controller development. For example, once the Model, View and Controller were developed, the process of implementing the physical system on the TINI board was relatively straightforward.

Figure 9. The physical implementation

The TINI prototyping environment provides a number of advantages and disadvantages for IEC 61499–based systems implementation. Advantages include low cost (both software and hardware), flexibility of the Java-based platform, and its support of multitasking and multithreading. Disadvantages include the board’s limited memory, real-time performance, and its execution model.

The TINI board’s support of multitasking and multithreading is certainly an important criteria for development of IEC 61499-based systems. For example, as Rumpl et al. [11] point out, this allows IEC 61499 function block applications to be executed in a “single tasking with multi-threading” environment. In other words, on each TINI board one thread would support the management services described previously; two more threads would support the input service interface function blocks (SIFB). These input SIFB threads allow the IEC 61499 application to react to external events such as changes in the sensor states and messages on the Ethernet connection respectively. Output SIFBs and other function blocks are triggered by the output events of the input SIFBs.

Although this execution model allows one to implement IEC 61499 applications, it does have its limitations. For example, parallel execution of basic function blocks and output SIFBs is possible if
different input SIFBs initiate their execution [11]. As well, it should be noted that this execution model will avoid data inconsistencies by synchronizing threads (i.e., by not allowing a thread to enter a function block that is blocked by another thread) and ignoring subsequent events while an event is executing.

The main drawbacks of the TINI board implementation are its limited memory and its execution speed. IEC 61499 class files are stored in non-volatile SRAM, which supports a maximum of 1 MB on current versions of the TINI platform. Both tasks and threads are scheduled in a round-robin fashion by the TINI operating system; the task scheduler is launched every 2 milliseconds and the thread scheduler is launched every 2 milliseconds [10].

A final limitation of the TINI board is its lack of support for dynamic class loading and serialization as noted previously. This is particularly relevant when investigating dynamic reconfiguration. Although it is possible to perform proof-of-concept tests of dynamic reconfiguration, this requires pre-loading all of the classes required for reconfiguration (i.e., a “contingencies” approach to reconfiguration is required) [12].

Although the system described in this paper is quite simple, we feel that this approach to developing real-time distributed control systems would be particularly useful for larger systems. In particular, the Model and View modules have the potential to be very useful for initial system verification and later to enable system reconfiguration (e.g., the addition or removal of physical devices or changes in the functionality of the system). Given that the Model and View modules are accurate, this approach could significantly reduce the time required to commission distributed control systems and also assist system upgrades. The effectiveness of this approach would be greatly assisted by the development of libraries of function blocks that could be used to develop the Model and View modules.

Our current work in this area is focused on the issues of fault monitoring and recovery in distributed control systems. One of the main advantages that this type of system provides is the opportunity to recover gracefully from failures of individual elements of the distributed system. Currently, we are looking at a “contingencies” approach to fault recovery where contingent function block code is instantiated within the distributed control system when a device fails (e.g., if the conveyor controller fails, the dispenser controller takes over its functionality). This approach requires planning in advance, or developing contingencies for possible failure modes. Given the advances in the area of distributed artificial intelligence however, our ultimate goal is to develop intelligent reconfiguration techniques that will allow controllers to reason about the current state of the system and develop strategies to overcome problems.

8. Acknowledgements

The authors wish to thank the Natural Sciences and Engineering Research Council of Canada and the University of Calgary, Faculty of Engineering for their generous support of this research.

9. References