An Extension to the Foundation Fieldbus Model for Specifying Process Control Strategies

EE382C: Embedded Software Systems, Spring 1999

Prof. Brian L. Evans
Department of Electrical and Computer Engineering
The University of Texas at Austin

Michael Schaeffer
May 8, 1999

Abstract: As the process control community moves towards open standards, efforts like the Fieldbus Foundation's Foundation Fieldbus will play an increasingly prominent role. This paper discusses an effort to integrate an imperative programming language into the dataflow programming model specified by Fieldbus.

Introduction

As society enters the 21st century, one of the more profound differences between now and the previous century is the important role of large scale chemical production facilities. These facilities are instrumental in making the refined chemicals needed for the production and usage of manufactured goods. Everything from aspirin to motor oil are made in immense quantities by automated production lines, and as our society has grown, so too have our demands on these production facilities. To meet this demand, a process control industry has emerged to help automate and control the processes of chemical refining and production.

A typical process control application is controlling the level of fluid in a reactor vessel. The process control system is composed of some form of programmable logic controller (PLC) and a collection of I/O sensors that interface it to the outside world. For controlling the level of fluid in a tank, there will be a pressure sensor to measure the fluid level and a valve to control the rate at which fluid flows into the tank. The industrial computer will sample the level sensor periodically, compare the actual value against the desired value, and calculate a new position for the valve, typically using the PID control algorithm. Currently, the control algorithm is usually implemented in one of the five IEC1131 languages, sometimes BASIC, and resides entirely on the PLC. The PLC monitors the level sensor and controls the valve using a pair of wires for each device and monitoring or controlling the current in the wiring loop between the device and the PLC. Two of the main disadvantages to this approach are a high cost to the wiring between the PLC and the devices, and a closed product architecture. Typically, vendors develop a product architecture that strongly favors staying within a given product lineup; this can lead to increased engineering costs for control solutions.

In an effort to solve some of the issues of process control, one of the more recent developments in the process control field is Foundation Fieldbus. Foundation Fieldbus is a specification that combines a networking protocol and an application model. The networking protocol defines a standard that allows intelligent devices to co-exist on one network. By defining a class of periodically scheduled traffic, the Fieldbus networking standard allows devices to communicate with each other in a predictable fashion in which a basic level of functionality is guaranteed. In addition to the network standard defined by Fieldbus, the Fieldbus specification defines an object oriented application model and a set of objects built using that model in which to build interoperable control systems. To provide a guarantee of interoperability, Fieldbus devices are formally tested against these specifications before they can be certified as an interoperable device.

This paper will introduce the application model of Foundation Fieldbus and discuss the development of an extension to that model. The design choices involved in making the extension will be discussed, along with the effects of the extension on the computational capabilities of Fieldbus. Finally, a few alternatives for future advancement of control system specification in Fieldbus will be discussed.

A Variant of Synchronous Dataflow

The fundamental object in a Fieldbus control application is the function block. Analogous to stars in the Ptolemy dataflow toolset [2], a function block is an actor in a dataflow network. Function blocks can contain input parameters that subscribe to information published by output parameters on other function blocks. The connections between input and output parameters are defined by another Fieldbus object– the linkage object. [3] The linkage object contains the information used to bind two

parameters together, including an optional network connection that can be used if the function blocks being linked exist on different devices.

Once this dataflow graph has been created, the devices containing the objects it is composed of must be configured to run the graph. Like the Synchronous dataflow model (SDF) the dataflow graph can be synchronized statically. [12] However, in stark contrast to the SDF model, for each execution, Fieldbus function blocks are constrained to produce or consume one data value per I/O parameter. This trivializes the computation of the repetitions vector (each node runs once) and thus trivializes computing a schedule to computing a topological sort of the dataflow graph.

The other significant departures from the SDF model are the notions of execution time and concurrency. One of the central tenets of deterministic control in Fieldbus is that the control application should run with predictable and consistent time behavior. To achieve this, the Fieldbus model requires each function block to have a declared maximum run-time. When scheduling the function blocks, they are scheduled relative to a time offset, rather than to an ordering. As an illustration, where an SDF schedule would read “fire block A and then fire block B”, a Fieldbus schedule would read “fire block A at offset 0ms, fire block B at an offset of 10ms and repeat this cycle every 25ms.”

As with block execution, communications on the Fieldbus also have a ‘declared’ upper bound on the time in which they will take. Because of the predictable nature of scheduled traffic on the Fieldbus, examining the communications parameters of the bus allows the calculation of an upper bound on the length of time it takes to transmit a parameter from one device to another. As a result of this, it becomes possible to take into effect communications between multiple devices as the schedule is calculated. A function block graph can then be scheduled on multiple devices, and the concurrency allowed by multiple processor elements can be used to decrease the overall execution

time of the graph. An important difference between the SDF model and the Fieldbus model is that processors are not heterogeneous in the Fieldbus model. Since a block might be performing physical I/O, that block must run on the device containing the I/O transducer.

Along with defining a framework in which function blocks may be defined, the Fieldbus specification defines a set of ten standard function blocks. These function blocks encompass basic I/O operations like reading an analog I/O channel as well as control functions, like the PID algorithm typically used in applications like tank level control or temperature control. For a device to be registered with the Foundation, every block based on one of the ten standard blocks must pass a rigorous test regimen. This allows the Foundation to say with confidence that the device will behave predictably when networked with other registered devices. To facilitate adding value to the standard Fieldbus function blocks, the Foundation allows blocks with extended sets of I/O parameters to be defined, as long as the default values for the device-specific parameters results in a block that will pass the test suite. The Foundation also allows for the presence of untested blocks. Custom blocks, like the block described in this paper, fall into this category.

The Expression Block

The ideal Fieldbus device would support all ten standard blocks with a large amount of manufacturer-specific functionality. This would enable customers of the device to utilize the full range of control defined in the specification. This model has two significant flaws: a relatively low return on the investment of developing all ten blocks, and the possibility that the set of ten standard blocks are insufficient to implement a given control strategy. To solve this, a manufacturer specific block with the

capability of calculating arbitrary expressions is a natural fit for the job. In addition to being lower cost to develop than a standard function block, the expression block encompasses a much wider variety of control than a standard fixed function block. I developed the expression block described in this paper for the National Instruments FP-3000, a Fieldbus compliant process controller. In addition to the expression block, the FP-3000 supports a set of Fieldbus standard function blocks for I/O as well as process control.

To implement an expression block, there are a number of engineering decisions that need to be made. The choice of expression language is one such decision. To minimize the effort involved in writing a language parser, as well as to use a well-established syntax, I chose an extreme subset of C. Unlike the full language, the language supported by the expression block only supports C expressions, "if...then...else" statements, and a collection of state variables that persist from one block execution to the next.

The second major engineering issue was the implementation of the expression engine. To guarantee that each pathway through the expression is valid, it was essential that the entire input expression be checked for syntax and other similar errors. This and the anticipation of future efforts to calculate a bound on the run-time of the expression resulted in the selection of a implementation of the expression language that uses a compiler and a byte-code interpreter. Each time the expression is changed, it is compiled into an intermediate bytecode for a stack-based machine. For each function block execution, the bytecode interpreter is invoked in the bytecode produced by the compiler. In this manner, the expression can be checked for validity as soon as it is downloaded and decomposed into a format that has a low execution overhead. In addition, the bytecode produced by the compiler offers an easy way to estimate an

upper bound on the execution time of the block. By assigning each bytecode its own execution time and summing up all of the individual bytecode execution times, an upper bound may be calculated. To compute more accurate bounds, the results of the Cinderella project could be applied to the calculation. [8]

The last design issue I discuss, and perhaps the most significant issue from the end-user's point of view is the handling of Fieldbus value/status pairings. When a data value is published from one function block to another, it carries with it a status, a description of the quality of the value being published. One use of this is to report a sensor failure. If a device detects a failure in the physical device hardware, it can report that fact to the function blocks that implement the control algorithm. The control algorithm can then initiate a fail-safe behavior to preserve the integrity of the process and prevent capital damage. In addition to carrying information about the quality of the data, the status also carries information describing the 'limiting' of the value. If an input sensor is at the upper or lower bound of its scale, the function block can report the value as high or low limited respectively. In the case of output devices, such as valves, the valve can report being fully open or closed to its controlling function block. This enables the controlling algorithm to behave predictably as the system approaches the limits of its behavior.

Unlike the standard function blocks, the expression block has no pre-defined status behavior. That is, the Fieldbus specification does not define a general means for deciding how a status is to progress through a series of calculations. As a result, I decided to adopt a simple set of rules and allow the user to override those rules with a set of functions intrinsic to the expression language. In the case of binary operators, the output of the operator carries the lowest status of the two input parameters. In a case where a 'good' value and a 'bad' value are added, the sum has a status of 'bad'. If both

summands are ‘good’, the sum is also ‘good’. In the case of the limit bits, I have defined, and documented, a set of rules, I believe to be sensible. In the case of addition, if the two summands are limited in the same direction, the sum is also limited in the same direction. If the summands are limited in different directions, the sum is not limited at all.

The rules described above are intended to allow end users to use the expression block in simple applications without concern for the status logic of Fieldbus. In the case where a linearization stage is needed between an input parameter and a control block, it is safe to assume that the quality of the linearized output is the same as the quality of the output from the input sensor. In this case simple status propagation is all that is needed. In the case where multiple input sensors are combined into one value, it is also safe to assume that the quality of the output is only as good as the worst quality input value. In cases where these assumptions are not true, or the user is sophisticated enough to control the status logic manually, the expression block provides a set of standard functions for controlling the status.

Futures

Like many other customer-driven ventures, the future of the work described in this paper is at the mercy of the people that have an interest in subsidizing development by buying the product. As a result, it is hard to accurately predict the direction in which the work will progress. Having said that, I believe that in the short run, ideas presented in this paper should be extended to be more standards oriented, allow more expressiveness, and aim for a high level of usability. In the long run, I believe that this work should be significantly extended into a general-purpose system for programming control strategies.

Taken to the limit, the idea of an expression block leads naturally to an architecture in which entire function blocks can be developed in an external environment and downloaded at run time to the Fieldbus device. Development of suitable tools would enable such a function block to be specified in a graphical programming environment and programmed using a standard, IEC1131, language. This would enable developers with existing PLC programming experience to migrate their skillset and existing code to the environment of function blocks on intelligent devices.

Accomplishments and Conclusion

This work has many logical precedents. Ranging from integration of languages like Matlab into Ptolemy [1], to a similar expression language embedded into National Instruments' LabView dataflow model [9], the concept of embedding a language into a dataflow graph is not new. The fundamental accomplishment of the work described in this paper is the potential extension of a product into a new set of problem domains. The FP-3000 as it initially shipped was suitable for use only as a PID loop controller, the most basic class of control device. With the introduction of this limited form of programmability, the FP-3000 has been endowed with the capability to implement arbitrary control logic. Thus, within the performance constraints of the product, it should be more able to compete, featurewise, against existing PLCs. The second accomplishment of the work described in this paper is the creation of an example, suitable for end-users, of heterogeneity in the SDF model of computation. The act of embedding an imperative language with simple control flow into a dataflow graph, and the ability to maintain state across executions is tantamount to embedding a finite state machine into that same dataflow graph.

References

1. B. L. Evans. “Matlab and the Ptolemy/Matlab Interface”, DSP Design Group Meeting, University of California at Berkeley, Berkeley, CA.

2. E. A. Lee. “Overview of the Ptolemy Project”, ERL Technical Report UCB/ERL No. M98/71, University of California, Berkeley, CA 94720, November 23, 1998.

3. Fieldbus Foundation, *Foundation Specification: Function Block Application Process: Part 1, version 1.3*, Fieldbus Foundation, Austin, Texas, 1998.

4. Fieldbus Foundation, *Foundation Specification: Function Block Application Process: Part 2, version 1.3*, Fieldbus Foundation, Austin, Texas, 1998.

5. Fieldbus Foundation, *Foundation Specification: Function Block Application Process: Part 5, preliminary*, Fieldbus Foundation, Austin, Texas, 1999.

6. K. Nielsen and W. Schmidt. “Performance of a Hardware-Assisted Real-Time Garbage Collector,” *Sixth International Conference on Architectural Support for Programming Languages and Operating Systems*, 1994, San Jose, CA.

7. K. Nielsen. “Issues in the Design and Implementation of Real Time Java,” *Java Developers Journal*, http://www.sys-con.com/java/iss1/real.htm

8. M. Jacome. “H/S Codesign: Performance Analysis of Software” http://www.ece.utexas.edu/~bevans/courses/ee382c/lectures/20_hwsw/cinderella.pdf

9. National Instruments, *LabView User Manual”* National Instruments, Austin, Texas, 1998.

10. R. Atherton. Moving Java To The Factory , *IEEE Spectrum*, December 1998, vol 25, no 12

11. R. Valdes. “Little Languages, Big Questions”, Dr. Dobbs Journal. September 1991

12. S. S. Battacharyya, P. K. Murthy, and E. A. Lee, “Software Synthysis from Dataflow Graphs”, Kluwer Academic Publishers, Norwell, Mass, 1996.

13. Sun Microsystems Inc. *The Java Language Environment*: A White Paper. 1995 Sun Microsystems Inc.: Mountain View, CA.

