Literature Survey:
Extending Real Time Dataflow with Arbitrary Logic

Michael Schaeffer
March 23, 1999

EE382C: Embedded Software Systems, Spring 1999

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

Abstract: As Foundation Fieldbus becomes more widely used for the development of process control solutions, the limitations of having a standard, fixed vocabulary of function blocks will become more obvious to control system engineers. To help mitigate this problem, I intend to develop a way to safely specify arbitrary control logic within the dataflow based framework of the Foundation Fieldbus Function Block Application.

Introduction

I intend to explore alternatives for incorporating arbitrary logic, specified in an imperative language, into an existing dataflow development environment. This is intended to help mitigate the problem that arises when the vocabulary of blocks supported by a given dataflow system is insufficient to specify a desired behavior. As dataflow systems are an established tool for specifying and modeling systems, this problem has been addressed by a number of people and organizations, and many examples of techniques that address this problem are described in the literature.

The particular dataflow system I intend to extend, the Foundation Fieldbus function block model (FBAP), is unique in that it is deterministic, runs on an embedded platform, and is capable of being distributed across multiple devices. A user of the FBAP model specifies a dataflow graph using a network configuration tool. She may then assign blocks in the dataflow graph to devices on her network of devices. The configuration tool then schedules the periodic execution of the device's function blocks as well as the network communication that occurs between the various function blocks. Because it is precisely scheduled the network can guarantee deterministic execution of the dataflow graph.

Objectives

Because of the widespread acceptance of dataflow as a modeling and programming tool, many other people have tried to solve similar problems in various

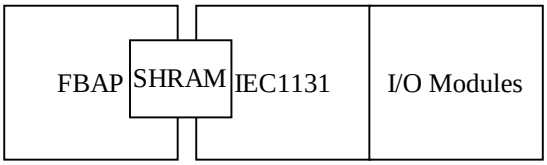
other dataflow systems. My primary intent in my literature survey was to explore pre-existing solutions for extending dataflow tools that might be useful applied towards extending the FBAP model. My secondary intent was to explore alternatives for the language in which the arbitrary logic was to be specified.

Extending Dataflow

As shown extensively by the Berkeley Ptolemy project, it is quite possible to bridge the gap between dataflow and other models of computation. Ptolemy, for example, contains what is perhaps the ultimate form of arbitrary block. In addition to blocks that can invoke Matlab and Mathematica expressions, Ptolemy has an editor in which blocks might be specified in a variant of C++. These definitions can be compiled into a shared library and linked into the environment seamlessly and at runtime. In a workstation environment, with relatively unlimited resources, this is a powerful and effective way to allow the user to incorporate arbitrary logic into a dataflow graph. However, in the embedded environment of Foundation Fieldbus devices, lower-impact solutions must be explored.

The Smar PLC's Foundation Fieldbus Interface

Smar, a Brazilian company specializing in Foundation Fieldbus process control solutions, has developed a scheme by which arbitrary logic may be included in a dataflow diagram. The Smar LC700 Programmable Logic Controller has the ability to run both FBAP dataflow diagrams as well as arbitrary programs written in IEC1131 ladder logic, an industry standard for programming process controllers. To bridge between the two development environments, Smar's device has a proxy block that allows FBAP dataflow applications to read and write to a region of memory shared with the IEC1131 logic interpreter. The LC700 has two seperate microprocessors, one is used to run IEC1131 code, and the other runs the FBAP dataflow graph. This hardware architecture is depicted below.



The Architecture of Smar's LC700

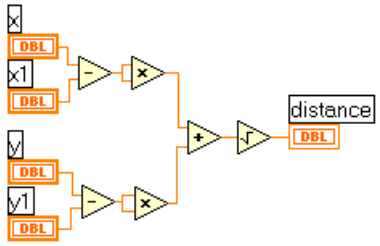
This implementation of programmable function blocks does run in an embedded device on a Foundation Fieldbus network. It also preserves the deterministic characteristics of the FBAP dataflow graph and allows interoperability with non-dataflow code written in an industry standard programming language. The difficulty with this implementation is that it does not place any timing constraints

on the code implementing the arbitrary logic. While the dataflow graph might be executing once a second, there is no guarantee that the logic behind the programmable function block is running at the same rate, if it is running at all. While this design allows the developer to specify arbitrary logic, it comes at the high price of a lack of determinism of the custom logic; the custom logic becomes a second class citizen.

The second difficulty with this architecture is that the logic behind the custom function block must be specified in terms of IEC1131. Where a FBAP dataflow arc contains detailed information on the quality of the data being transmitted, IEC1131 does not have the concept of quality information. This prevents fieldbus quality information from being seamlessly processed and effectively forces custom logic in a control system to become an opaque barrier for this quality information .

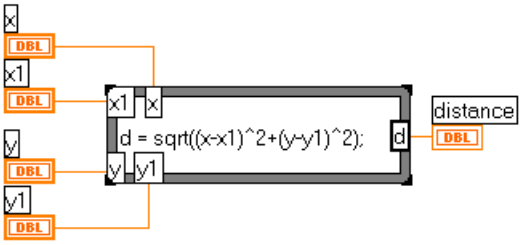
The LabView Formula Node

Probably the most well known dataflow tool is National Instrument's LabView. LabView was initially developed by National Instruments in 1986 to make it easier for non-programmers to develop software for a companion line of interface hardware. To ease the transition for engineers that might be accustomed to wiring diagrams, LabView adopted a dataflow model that allows VI's, the LabView name for a computation node, to be wired together graphically. A LabView program that calculates the distance between two points is shown below.



Dataflow Expression in LabView

One of the difficulties that LabView faces with the pure dataflow model is readability. Simple numerical expressions, like the formula shown above, become more difficult to read and extend than the textual equivalent. To help deal with this issue, LabView incorporates a formula node that allows numerical expressions to be entered textually. With this formula node, the distance calculation program shown above can be expressed as shown in the following picture. In this manner, LabView allows arbitrary numerical expressions, in a textual format, to be seemlessly incorporated into a dataflow graph.



The NI Function Block Shell

To support the development of blocks for Foundation Fieldbus devices. National Instruments has developed a tool, the Function Block shell, that provides an easy environment for developing function blocks. The function block shell allows C code to register blocks in a common database residing on the device. Once registered in the database, the shell will invoke callbacks to inform the user code of processing that needs to be done. As the block needs to be executed, the shell will invoke a callback, cbExec, that should contain the processing code for the block. The shell will also provide callbacks to inform the user level code of configuration changes and requests from the configuration utility to create and destroy instances of blocks.

The function block shell does effectively allow for custom blocks to be defined. In fact, in National Instruments' FP-3000 device, all of the blocks in the device are defined by writing code to the shell API. The difficulty with the shell is that it requires that the code that defines blocks be statically linked in the device's firmware. Since end users can not build the firmware image for the device, this keeps them from being able to define function blocks using the shell.

Domain Specific Languages

The other aspect of my investigation related to domain specific programming languages. There are a number of clear disadvantages to using a general purpose language, like C or C++, to specify arbitrary logic for a block in a dataflow diagram. To address these issues, I investigated a variety of 'little languages'. A 'little language' is a small, focused programming language, often embedded in a larger software system to solve a specific problem. One such example is the formatting language used by the C printf standard library function. It allows concise specification of a very particular type of problem.

In the case of adding arbitrary logic to a Foundation Fieldbus function blocks, the problem domain is the specifications of expressions for numerical computation. This is slightly complicated by the fact that the standard Fieldbus data types for data flow arcs contain a value and a status. The status is a description of the quality of the data being broadcast on the dataflow arc. It also contains information pertaining to the reason the quality is what it is as well as information describing if the value is limited or constant and unable to be changed. For an arbitrary logic block to effectively interoperate in a Fieldbus FBAP dataflow graph, it must manipulate and propagate the status information as it performs

calculations. Ideally, this would be as transparent to the user as possible.

The second requirement is that the user should be restricted from specifying logic that is of unbounded execution time. This is to be able to guarantee that a system will always work in a predictable manner and never cease to control the process due to a logic error in the specified control algorithm. A domain specific language makes it possible to have very precise control over what the user is allowed to do and not do and can greatly ease this problem.

While most existing examples of domain specific languages are too domain specific to directly apply to adding arbitrary logic into FBAP dataflow graphs, there are a number of general concepts that do directly apply. One common implementation of domain specific languages is that they should be compiled into a bytecode. The runtime engine becomes a simple bytecode interpreter and is not responsible for syntax checking and other forms of formal verification. This not only allows the specified logic to execute more quickly, but it allows errors in the program to be detected more quickly and reliably.

Conclusion and Goals

For the remainder of this project, I intend to develop a expression compiler and bytecode runtime engine that may be incorporated into arbitrary fieldbus function blocks. I intend to demonstrate this runtime engine acting both as a component of an existing function block as well as to implement a function block that implements arbitrary logical expressions. I also intend to demonstrate applications for this control block, including PID control, thermostat control, alarm condition detection, and discrete sequencing.

References

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

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.

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

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

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

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.

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

National Instruments, "LabView User Manual", National Instruments, Austin, Texas, 1998.

R. Atherton. Moving Java To The Factory , *IEEE Spectrum*, 25(12), December 1998

R. Valdes. Little Languages, Big Questions, *Dr. Dobbs Journal*. September 1991

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

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