Flow-Based Programming

J. Paul Morrison
J.P. Morrison Enterprises Ltd., Unionville, Ontario, Canada

Regular Technical Paper
Journal of Application Developers’ News

Abstract - In this paper, we give a short description of an application development technology called Flow-Based Programming (FBP), briefly describing its basic concepts, history, and some of our experience with it. An early implementation of FBP has been in continuous production use for almost 40 years, but it is attracting attention worldwide as it is a significant and important paradigm in building and maintaining computer applications.

Keywords: dataflow, agile, asynchronism, visual, componentry, maintainability

1 Introduction

Flow-Based Programming (FBP) [1] is a particular form of dataflow programming based on bounded buffers, information packets with defined lifetimes, named ports, and separate definition of connections. The web site describing it can be found at http://www.jpaulmorrison.com/fbp, which also contains links to other material about FBP, as well as to the FBP Google group (quite active, with worldwide membership) [2]. There is also an extensive article on Wikipedia [3]. A diagramming tool, called DrawFBP [4], is also available on the FBP site - it supports hierarchical diagrams, and, given enough information (names of components and ports), can generate running Java, C# and/or NoFlo programs (NoFlo [5] is an FBP implementation built on Node.js, an asynchronous JavaScript platform [6]). Additional languages can easily be added as required.

2 Concepts

The FBP development approach views an application not as a single, sequential, process, which starts at a point in time, and then does one thing at a time until it is finished, but as a network of asynchronous processes communicating by means of streams of structured data chunks, called “information packets” (IPs). In this view, the focus is on the application data and the transformations applied to it to produce the desired outputs. The network is defined externally to the processes, as a list of connections which can be interpreted by a piece of software, usually called the “scheduler”.

The connected processes communicate by means of fixed-capacity connections. A connection is attached to a process by means of a port, which has a name agreed upon between the process code and the network definition. More than one process can execute the same piece of code. Ports may either be simple, or array-type. If a connection becomes full, processes feeding it will be suspended; if it becomes empty, the process (there can only be one) will be suspended. It is the combination of ports with asynchronous processes that allows many long-running primitive functions of data processing, such as Sort, Merge, Summarize, etc., to be obtained “off the shelf”, and combined with application-specific components. All of these can be thought of as software black boxes. Fig. 1 shows a very simple network, comprising 3 processes.

![Fig. 1. Simple network of 3 processes](image)

In Fig. 1, A, B and C are processes executing code components. O1, O2, and the two INs are ports connecting the connections M and N to their respective processes. It is permitted for processes B and C to be executing the same code, so each process must have its own set of working storage, control blocks, etc. Whether or not they do share code, B and C are free to use the same port names, as port names only have meaning within the components referencing them and at the network level.

M and N are what are often referred to as “bounded buffers”, and have a fixed capacity in terms of the number of IPs that they can hold at any point in time.
Information Packets (IPs) are usually structured chunks of data. At any point in time, a given IP can only be "owned" by a single process, or be in transit between two processes. Some IPs, however, may not contain any real data, but are used simply as signals. A type of signal IP is a "bracket IP", which can be used to group data IPs into sequential patterns within a stream, called "substreams". Substreams may in turn be nested. IPs may also be chained together to form "IP trees", which travel through the network as single objects.

The system of connections and processes described above can be "ramified" to any size. During the development of an application, monitoring processes may be added between pairs of processes, processes may be "exploded" to subnets, or simulations of processes may be replaced by the real process logic. FBP therefore lends itself to rapid prototyping. Because the relation between processes is cooperative, rather than hierarchical, processes can easily be moved from one program or application to another, or even from one machine to another. Processes can easily be multiplexed to take advantage of multiple processors on recent computers, or to overlap I/O and communications functions with processing or other I/O. Fig. 2, generated using DrawFBP, shows 3 instances each of 2 components, running asynchronously, driven by an off-the-shelf load-balancing process (this is described in more detail in the book Flow-Based Programming, 2nd Edition [1]. This example just uses dummy components for demonstration purposes.

In the Java and C# implementations of FBP, multiplex numbers can be computed at network setup time, based on various criteria.

Implementations of FBP may be non-preemptive or preemptive - the earlier implementations were non-preemptive (IBM mainframe Assembler Language and C language).
whereas the later Java and C# implementations are preemptive. FBP programs naturally multithread, to take advantage of all the processors on a machine, without any of the normal headaches facing developers trying to do multithreading using conventional technology.

This is really an assembly-line image of data processing: the IPs traveling through a network of processes may be thought of as widgets traveling from station to station in an assembly line. “Machines” may easily be reconnected, taken off line for repair, replaced, and so on. Oddly enough, this image is very similar to that of “unit record equipment”, widely used to do data processing before the days of computers.

FBP provides a consistent application view from the highest level (companies, departments, individuals) to the lowest (application code), and has proven to provide a medium that enhances communication between analysts, developers, users, and operations personnel.

FBP lends itself to rapid prototyping, and so may be called an “agile technology”, but our experience over the last 40 years suggests that an even more important aspect is in the area of maintainability – an area that gets very little attention in the literature, but which is of paramount importance for companies, which typically spend 80-90% of their resources today maintaining old applications.

3 History

FBP was invented/discovered by J. Paul Morrison in the early 1970s, and an early implementation of this technology (called AMPS) has been in continuous production use at a major Canadian bank since that time, amounting to almost 40 years (one major program, processing some 5M accounts daily, went live in 1975). The late IBM architect, Wayne Stevens, wrote several articles describing and supporting the FBP concept [9], and included material about it in several of his books.

Mike Beckerle, currently at Tresys Technology, has experience using FBP for very large amounts of data, and has contributed several sections to the 2nd edition of my book, “Flow-Based Programming”, published in 2010, and available in paperback, epub and Kindle formats [1]. There he describes FBP-based solutions for a number of the problems involving in managing large amounts of data, and an approach to checkpointing long-running systems, which he developed with Sebastian Ertel.[10].

Henri Bergius of Nemein, Finland, has an interesting FBP implementation called NoFlo [5] built on Node.js [6], which has been attracting attention at JavaScript conferences. He recently demonstrated a small FBP application for touch screens. NoFlo is also supported by the DrawFBP diagramming tool.

The number of systems that implement some or all of the basic FBP concepts has been growing steadily; IBM currently sells a tool for general data transformation called DataStage [11] which combines FBP with parallel processing. A recent impressive implementation is called Storm [12]. FBP-like concepts are also showing up in the music and art world.

Unfortunately, much of the corporate usage of FBP is covered by confidentiality agreements, so it is hard to discover how widely it is being used. However, a number of companies such as InforSense [13], and Accelrys [14], as well as open-source Pentaho [15] and Knime [16], are mentioning or have mentioned FBP in their advertising. Other implementations that have shown up recently in the public domain are: PyF [17], DSPatch [18], Pypes [19], and NoFlo [5], referenced above.

I am including one more diagram, to try to convey the level of parallelism that FBP provides – Fig. 3 shows a fairly complex batch program, showing multiple I/O processes, and illustrating the “subnet” notation.

This program is an example of a very common type of batch program called an “Update”: a file of detail records is run against a file of master records - this matching is handled by an off-the-shelf component called “Collate”. The merged stream produced by Collate is then processed in various ways to produce a number of output files and error reports. With conventional programming, this type of program was extremely hard to write – with FBP it is quite straightforward. It is also worth pointing out that the FBP technology applies equally well to batch and on-line applications – the traditional distinction is no longer necessary!

Blocks in Fig. 3 with double boundaries represent “subnets” - these can be thought of as networks with one or more “sticky” connections.

4 Conclusions

FBP has proven over the years to be a very robust, productive technology for developing applications, yielding reliable, high performance, and highly maintainable systems in many different application areas. It exploits the developer's visual cognitive skills, while supporting hierarchical stepwise development. Applications built using this technology have run reliably for long periods (one for close to 40 years), while undergoing continuous modification due to changes in hardware, software, business and legislative environment. At the same time, FBP addresses many of the problems bedevilling application development today, appears well-adapted to high-performance, high-volume data applications, and also appears to tie into some of the most advanced thinking going on today in the area of distributed systems, and perhaps even hybrid software-hardware approaches.
5 References


