A visual parallel programming language

DataFlow

by **Gustavo A. Fernandez**
ARPAnet **FERNANDEZ@SU-SCORE**
CompuServe **71425,1456**

December 11, 1984

Introduction

DataFlow is the most Macish programming language written to date. The mouse is used for all operations except typing in numbers. DataFlow programming is a cross between flowcharting and electronic circuit design. While symbols similar to common flowcharts are used, the program behaves more like an electric circuit where numeric 'tokens' replace current going through wires.

DataFlow version 0.0 was designed mainly to show the concepts and shortcommings of Dataflow rather than as a useful tool. For this reason, many important features such as subroutines, arrays, structures, strings, and even load/save were not implemented. Large high-performance computers have been designed to run DataFlow, however. These machines have an architecture completely different from the Von Neuman architecture used by the the vast majority of machines today. If you are interested in DataFlow, the February, 1982 issue of **COMPUTER** magazine was devoted to this subject. This can be found in any good library which receives computer science journals.

Overview

When the program starts up initially, the user is presented with three windows on the screen: An icon window from which various menu selections are chosen, a field window where most of the work takes place, and an output window where program results are written.

The icon window is a vertical bar of 9 icons. The top two are labeled STOP and GO. They stop and start the machine respectively. Either one of these is highlighted at all times. Below these is a set of seven more icons, one of which is highlighted. The first is a tools icon which contains scissors, a hypodermic needle, and a toilet used to cut objects off the screen, inject initial input values, and flush all input queues respectively. The next icon is a wire which is used to wire objects together to establish a logical connection between two objects, or even the input and output of the same object. The remaining five icons are used to create objects. Each icon represents a different class. They are, from top to bottom: constants, arithmetic operations, comparison operations, input/output and select/distribute.

The Icon bar works very much like the menu bar laid on its side. Some of the icons have additional selections which pop to the right when the mouse is clicked on the icon. The last selection chosen is always showing. To select a different icon, click on the leftmost icon and drag right to the icon you want.

To put an object on the screen, first select it on the icon bar, and then click in the field window. Select the scissors and click on an item to remove it. You will notice that each object has small rectangles called **handles**. These are where you connect wires to. Handles are either for input or for output. Any number of wires can be attached to an output but only one wire can be tied to an input handle. Input handles are always on top or on the side of the object while output handles are always under the object.

To connect two objects, choose the wire icon and drag from an output to an input. If you are still dragging and find that you don't like the wire that you have drawn, retrace your steps and the wire will shrink automatically. The mouse must be near an input handle in order for the wire to become permanent. To delete a wire, select the scissors and click anywhere along the wire.

Types of Objects

The first object is the constant object which has a rectangular shape. Click inside and use the standard Mac keyboard editing features to enter in a 16 bit signed integer constant. Constants are special tokens which are never used up.

The simplest and most familiar objects are the arithmetic objects which all have a circular shape. +, -, x, ÷, and MOD are implemented. The lefthand input is at 10:30 and the right-hand input is at 1:30. The output is at 6 o'clock. An output token is fired when an input token is received on both inputs. All arithmetic is done on 16 bit signed integers. Overflow is not flagged. Divide by zero returns a large meaningless number but does not cause an error.

Below the arithmetic objects are the comparison objects: =, ≠, >, < ≥, ≤. which have diamond shapes. Each takes inputs just like the arithmetic objects and outputs a 1 if the comparison holds true or a 0 if it is false.

Input and output objects are next. The input icon is a rectangle on top of a triangle pointing downward. Inside the rectangle, input a numeric value in the same way as for a constant object. click in the triangle to fire a token with the value currently in the rectangle. The output object looks like the input object up-side down. It has no output handles. Instead, anything received on its input handle is printed both in the rectangle and in the output window if that window is highlighted. (on top).

The last two objects, select and distribute, implement IF/THEN functions. Both have a control input on the left-hand-side which is tested for 0/non-0. This is usually attached to the output of a comparison object. The select object has two data inputs and one output. If the control input is not equal to 0, then the left hand input is fed to the output, otherwise, the right hand input is used. The distribute object has one data input and two outputs. If the control input is non-zero, then the input is sent to the left output, otherwise, the right output gets the token. Select and distribute are almost always used in pairs as described later in the examples.

Tokens and queues

A **token** is the basic item of data used to communicate between objects. In this implementation, only 16 bit signed integers are supported. All inputs to an object must have tokens ready in order for that object to fire. The one exception is that the input which is not read in the select object need not have a token ready. It is possible for more than one token to arrive at one input before any have arrived at the other inputs. In this case, the object cannot yet fire its output, and the input tokens must be placed on a queue and await their turn. Each input has a maximum queue size of five. A token will not fire unless all of the queues of the inputs attached to its output are not full. If any of the queues are full, a chain reaction will usually take place throughout the machine so that no tokens fire. This is called a lockout and indicates a bug in your program.

All input queues can be flushed (emptied) by selecting the toilet icon from the tools set which also contains the scissors. In this same set is also the hypodermic needle used to manually insert tokens into an input queue. This is a common technique used to set starting conditions for loops, etc.

Examples

The first example is a program to generate an iterative fibonacci sequence. It requires but one addition object and one output object as shown below: