Using Khoros/Cantata as a Visual Programming Language for Process Simulation

Harvey A. Cohen and Dougal J. Beilby
Computer Science and Computer Engineering
La Trobe University Bundoora Victoria
Australia 3083
Email: H.Cohen@latrobe.edu.au

Abstract

The paper describes experience in using the Visual Programming Language Cantata in the implementation of a specific process control simulator - a Digital Logic Simulator. Cantata is the VPL component of Khoros, a comprehensive image processing system developed for the X-Windows environment, has extensive image processing capabilities for graphic and imagery rich simulations Although Khoros/Cantata, which is based on a very highly specified data-flow model for image processing has no actual state-transition handling scheme, we found that propagating a 'common clock' to all processing elements enabled a proper sequencing of sequential processing.

Keywords: Visual Programming Language, Khoros, Cantata, Simulation,

Simulation and Khoros/Cantata

This project involves the development of a visual programming system for simulating systems that are composed of stable elements with interconnections and communications. The

system is to provide a graphical user interface that will allow the user to program by drawing a visual representation of the system to be simulated on the computer screen. Systems that are composed of stable elements with interconnections and communications, are inherently pictorial, and as such make a diagraming based system of visual programming the obvious choice. In the realm of image processing just such a visual programming scheme has been developed by Ransmure et al at the University of New Mexico. It is a large system by current standards - being some 60 Meg in size, with the actual visual programming sub-system called Cantata. Khoros is well documented and highly extensible. It is based on a data flow model for image processing. Cantata has the hooks for programming this data-flow model, through use of widgets called glyphs - which feature input and output buttons, whose interconnection by the user clicking with a mouse directs data flow. In this paper, for clarity, we refer to Khoros/Cantata rather than Cantata. Our implementation of the digital logic simulator in Khoros/Cantata, where the focus is on the state sequence of logic gates, demonstrates that such a data-flow model can be purposefully used for process simulation.

Process flow Modelling.

A simple sequential process has the form:

Initial ---> [Process 1 at time t] ---> [Process 2 at time t] ---->Final Output

where the arrows denote the flow of output at each stage. More elaborate systems have multiple outputs and parallel processes and even feed-back of outputs. In a simulation of such process flow one wishes to display the state of each process at a given time. Digital logic circuits modelling sequential counter logic, as embodied in counter operations, have precisely this process structure, and so provide the simplest instance of sequential processes for credible simulation. On the other hand, a simulation based on Khoros/Cantata must use the data flow model implicit in image processing:

Initial_Image ---> [Process1] ---> [Process2] ---> Final Image.

The Cantata widgets - glyths - have buttons whose visual interconnection direct the corresponding data flow. In fact the scheme is that temporary files are created corresponding to each

link, so that - for a simple examples like that drawn with no branches or loops -- implementation is in the form of Unix-Pipes:

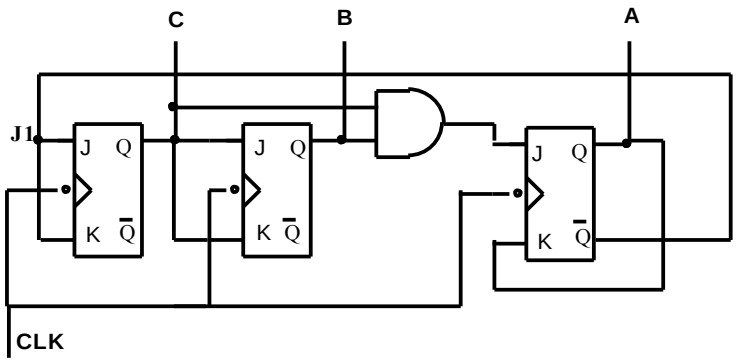
Initial_Image|Process _|Process_2|Process_3>Final_Image

where Initial_Image and Final_Image are file names, and the pipe | denotes file transfer.

Simulation of Digital Logic

Digital logic involves just a few stable devices, logic gates and flip-flops, which have for the specialised user well-known graphical forms. The on-screen construction of simple circuits via selection of these processing elements, and their interconnection, provides a rapid means of producing most sequential logic circuits, In sequential logic circuits, one is concerned with the sequential behaviour of all circuit elements, following a sequence of inputs. In a digital logic simulator once a circuit has been generated on-screen the user can view the sequence of logic states for all elements for some chosen input sequence.

To clarify matters, a simple circuit involving three JK flip-flops and one AND gate is shown:



MODULO 5 COUNTER

Certain digital outputs of interest in a simulation are labelled. However, what is vital for our purposes is the common signal to the flip-flops - a timing signal. In our simulator implementation the timing signal is propagated to all processing elements, not just to those where the common clock serves it usual role in synchronization.

By conceiving of the common clock in this way a sequence of time states can be stepped through in the simulation. It is notable that in many published circuits the actual common clock line is not shown. This is actually a desirable situation for a simulator, as it eliminates excessive clutter of lines whose interconnection is required.

Conclusions.

Khoros/Cantata is based on a simple data-flow model, - with no explicit state-transition handling capabilities (Jacob 1985) and not really intended for displaying state evolution for interacting processes. Nevertheless we have been able to implement a digital logic simulator using Khoros/Cantata. Our methodology, involving use of a common clock input for all processing units, not just those 'really' connected by a common circuit clock,, means that a sequence of 'time instants' is an input to all elements: this digital logic strategy can be applied to general simulations. Thus Khoros/Cantata can usefully function as a VPL for arbitrary process simulations. It is notable that Khorus/Cantata - is supplied to educational institutions at a nominal cost - - but boasts a extremely large library of image handling routines. We have not had an opportunity to make a comparison with the better known general purpose graphical toolkits such as Motif and InterViews (Linton et al 1988)

References

S. Chang, (1990) Principles of Visual Programming Systems, Prentice Hall New Jersey.

S. Cook et al (1989) Visual Programming of User Interfaces, in A. Kilgour (editor), *Graphics Tools for Software Engineers*, Cambridge, Cambridge University Press, pp. 103-112.

R.J. Jacob (1985) A State Transition Diagram Language for Visual Programming, *IEEE Computer*, 18(8) 51-59.

M.A. Linton, P.R. Calder, J.M. Vlissides (1988) InterViews: A C++ Graphical Interface Toolkit, Technical Report CSL-TR-88-358, Stanford University.

R.V Rubin , EJ Colin SP Reiss ThinkPad: A Graphical System for Programming by Demonstration, *IEEE Software*, March 1985, pp. 73-79.

.