

Cantata : Visual Programming Environment for the Khoros System

Mark Young · Danielle Argiro · Steven Kubica

Khoral Research, Inc. *

March 16, 1995

Abstract

Cantata is a visual programming environment within the Khoros system. Cantata contains many features not typically found in visual programming environments such as visual hierarchy, iteration, control structures, expression-based parameters and program encapsulation. This paper presents an overview of Cantata and these features.

1 Introduction

Khoros is a software integration and development environment which includes a suite of software development tools along with an extensive collection of image processing, data manipulation, matrix processing, and scientific visualization applications [1]. The use of a graphically expressed data flow visual programming environment called Cantata within the Khoros system allows users to easily bring together these numerous applications and have them work together as a cohesive whole.

Visual programming environments provide graphical or iconic elements which can be interactively manipulated according to some specific spatial grammar for program construction [2].

In Cantata [3], visual programs are created as directed graphs, where each node of the graph is an iconic element representing a program and each directed arc represents a path over which data flows. By connecting the data paths between programs user can interactively draw out a solution in a more natural way that matches their mental representation of the problem. By providing a visual environment for problem solving, Cantata increases the productivity of both researchers and application developers, regardless of their programming experience.

*<http://www.khoros.unm.edu/>

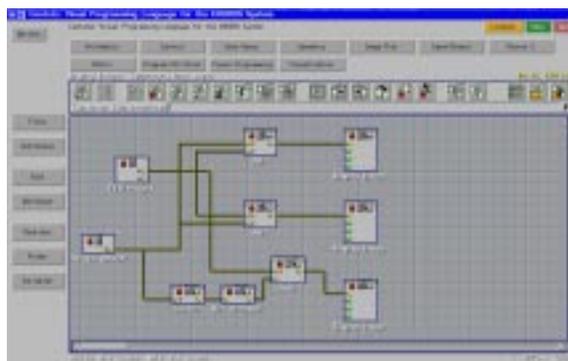


Figure 1: Illustration of Cantata running a visual program.

In Cantata, the icons called *glyphs* represent programs from the Khoros system. Each of the hundreds of stand-alone data processing and scientific visualization programs in the Khoros system can be represented in Cantata as glyphs. When accessed in the visual language, a Khoros program is referred to as an *operator*. To create a visual program, the user selects the desired operator, places the corresponding glyphs on the workspace, and connects the glyphs to indicate the flow of data from operator to operator, forming a *network* within a *workspace*.

Control structures can be used to branch and merge data flow, or to implement loops. Workspaces can be executed, saved and restored to be used again or modified later. Workspaces may also be encapsulated into independent applications with a very simplified graphical user interface so that they may be treated as stand-alone Khoros applications.

Visual hierarchy, iteration, flow control, and expression-based parameters extend the data flow paradigm to make Cantata a powerful simulation and prototyping system. Data and control-dependent

program flow is provided by *control structure glyphs* such as if/else, while, count, and trigger. Visual sub-routines, or *procedures* are available to support the development of hierarchical data flow graphs. Variables may be set interactively by the user, or calculated at run time via mathematical expressions tied to data values or control variables within the visual network.

2 Data/Control Connections

Glyphs contain input and output *data connection nodes*. When two glyphs are connected with a *data connection*, it is implied that the output of the first will become the input of the second; thus, the data connection represents data flow in the visual program.

A visual program requires data connections between glyphs in order to form the network and to define where each process will obtain its data. In contrast, *control connections* are not necessary as part of a fully operational network. They do, however, allow one to constrain the operation of a visual program and provide additional control over the order in which processes are executed.

Glyphs contain input and output *control connection nodes* which cause the second, or *controlled* glyph, to “wait” on the execution of the operator represented by the first glyph. Control connections provide a simple way of specifying an order for process execution when one is not already dictated by the data flow, as is frequently the case in networks with a number of parallel paths.

3 Variables

Variables can be defined and expressions utilizing those variables can be evaluated within Cantata. Once defined, variables and expressions can be used in place of integer, float, and double arguments of glyphs. Valid expressions may include variables, standard arithmetic operators, and logicals, as well as predefined constants and functions. The use of variables and expressions is required by many control structures.

4 Control Structures

To be a true programming language, a [visual programming] system must include the ability to handle conditionals and iteration [4][5].

Control structures allow one to create more complex visual programs by providing constructs that can be used to direct the flow of your visual program. In this context, “flow” refers to both “control flow” and “data flow”. When there is more than one possibility for data flow, control structures are used to determine which path the data flow will take.

Control structures fall into two major categories: *looping constructs* and *conditional constructs*. Looping constructs cause control flow of the visual program to iterate through a particular section of the network multiple times [6]. Looping constructs depend on the value of variables to determine how many times the control flow will iterate on the specified section of the network. The available looping constructs are:

Count Causes the data flow to iterate through a particular section of network a specified number of times.

While Causes the data flow to iterate through a particular section of network as long as a particular condition is met.

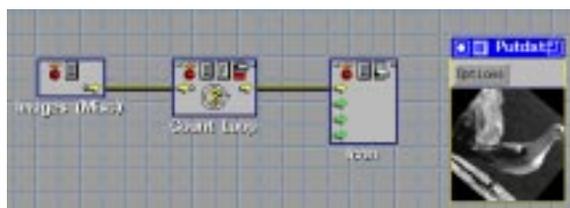


Figure 2: A visual program utilizing a count loop glyph to repeatedly rotate an image.

In contrast to looping constructs, conditional constructs imply that control flow of the visual program will be directed one way or another, depending on the value of variables defined for the workspace. The available conditional constructs include:

If/Else Directs data flow to one path if a specified condition is met, to another path if the condition is not met.

Merge Directs data flow from two separate paths to the same path, whether data arrives from the

first path, the second path, or both; no condition is involved.

Switch Selects one of two inputs for use by the visual network, depending on the value of a conditional statement.

Trigger Delays execution of a glyph until data is made available by another glyph which is not otherwise connected to the dependent glyph.

Expression Sets the value of a variable; used in conjunction with other control structures for purposes of variable initialization and/or modification.

Break Used in conjunction with looping constructs to direct data flow to terminate the running of a while or count loop which will cause the glyphs downstream from the loop to be executed.

Continue Used in conjunction with looping constructs to direct data flow to continue the next iteration of a while or count loop.

The values of defined variables may be changed by directly by the user from a variables pane or by an operator from an expression glyph. Looping constructs have built-in mechanisms for variable value update.



Figure 3: A visual program utilizing control glyphs.

5 Procedures

Hierarchy is supported within the Cantata visual language in the form of procedures. Similar in concept to a subroutine in a textual programming language, a visual programming procedure allows you to modularize a visual program so that certain operations are confined to a particular location within the visual program. Procedures promote readability of a visual program. In addition, they allow a certain portion of a network that performs a particular function to be used multiple times within a visual program. The use of procedures is especially effective with large and complex workspaces, as well as with workspaces that

are required to perform a particular task a number of times.

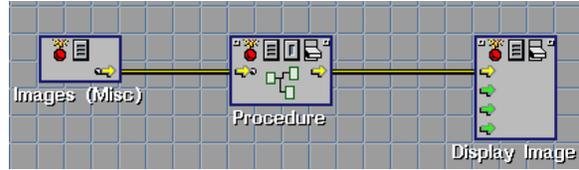


Figure 4: The procedure has been created to contain the “Shrink” and “Rotate” glyphs.

When the procedure workspace is open the main Cantata workspace, displaying the “outer” visual program, will be replaced with the procedure workspace, displaying the network that comprises the procedure. If desired, procedures may be nested.

6 Visual Network Scheduling

Cantata interprets the visual network dynamically to schedule glyphs and then dispatch them as processes. The Cantata scheduler is event driven, rather than data driven or demand driven. Once a glyph has been scheduled, the dispatcher is responsible for determining the data transport, the communication protocol, and the process execution mode. The data transport between the glyphs can be permanent (using files or shared memory) or non-permanent (using sockets or streams). The communication protocol between Cantata and the different glyphs can be as simple as initiating process execution, but may be more complicated if glyph parameters must be continuously updated as the process executes continuously. Glyphs may be executed locally or remotely to efficiently utilize a heterogeneous network of computers.

7 Encapsulated Workspaces

Once a visual program has been completed, it may be used to create an independent application that can be run without display of the network of glyphs that defines the visual program. The functionality embodied by the visual program may be utilized as desired, independently of the Cantata visual programming language. The process of creating such an independent application from a visual program is referred to as encapsulating a workspace, because one is “encapsulating” the functionality of the network as it exists in

the Cantata workspace, and “storing” it. The resulting program is called an encapsulated workspace.

Another important use of the encapsulated workspace is that it be brought into Cantata as a new operator just like any other Khoros program. Often, visual programmers will create encapsulated workspaces from visual programs simply in order to have them accessible as independent programs; such re-usability may even outweigh the capability for independent execution as a reason for encapsulating workspaces from visual programs.

Once a visual program has been encapsulated, the resulting program can be run directly from the command line, or accessed via Cantata as a new operator. Like any other program in the Khoros system, it can be edited and maintained within the Khoros software development environment.

An encapsulated workspace is not composed of either a C program or a scripting language interpretation of the visual program. In fact, the key components of an encapsulated workspace are:

1. The network of glyphs as they appeared in the Cantata workspace when the encapsulated workspace was created, written out as a saved workspace file, just like any other saved workspace file.
2. The user interface description file which simultaneously defines the workspace graphical user interface and the command-line user interface of the encapsulated workspace.
3. An automatically generated shell script which provides the command line user interface to the encapsulated workspace.

Creating an encapsulated workspace from a completed visual program is a convenient way to allow others (who may not be familiar with visual programming) to use the visual program without being concerned, or even aware, of its origin. Because a program created from an encapsulated workspace need not display the graphical user interface of the visual program, it can also be run in “batch” mode; for example, the program could be executed repeatedly by a script that could be run overnight.

8 Conclusion

By combining a natural environment of visual constructs with the programming features typically

found in textual languages, Cantata provides a powerful problem solving and prototyping system. Visual hierarchy, iteration, flow control, and expression based parameters augment the traditional data flow paradigm so that Cantata can be used effectively in a number of domains, including process control, simulation, and system integration. When combined with the data processing and data visualization programs available in the Khoros system, Cantata is particularly well suited for scientific data processing and visualization.

The flexibility of the visual network scheduler allows this single visual programming environment to simultaneously address the needs of these diverse domains. Furthermore, the visual programming paradigm is enriched by the ability to combine operators from different domains as desired in a single visual program, allowing the visual programmer to solve more complex and diverse problems than would otherwise be possible.

References

- [1] What is khoros? home page describing khoros 2.0, <http://www.khoros.unm.edu/khoros/khoros2/home.html>. 1995.
- [2] E. J. Golin and S. P. Reiss. The specification of visual language syntax. 1(2):141–157, 1990.
- [3] Khoros visual programming manual, khoros manual: Volume iv – khoros 2.0 manual set. 1995.
- [4] B. A. Myers. Taxonomies of visual programming and program visualization. 1990.
- [5] A. L. Ambler and M. M. Burnett. Visavis: a higher-order functional visual programming language. 1(2):159–181, 1990.
- [6] A. L. Ambler and M. M. Burnett. Visual forms of iteration that preserve single assignment. 1(2):159–181, 1990.

Khoros is available via anonymous ftp from <ftp.khoros.unm.edu> or via www on <http://www.khoros.unm.edu>. Khoros is a copyright of Khoros Research, Inc.