Using Tcl to Control a Computer-Participative Multimedia Programming Environment *

Christopher J. Lindblad †

Telemedia Networks and Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology

Abstract

This paper describes how the VuSystem, a programming environment for the development of computer-participative multimedia applications, is controlled through Tcl scripts. In the VuSystem, networks of in-band media-processing modules are created and controlled by interpreted out-of-band Tcl scripts through object commands and callbacks. Tcl's extensibility, simple type system, efficient interface to C, and introspective capabilities are used by the VuSystem to produce a highly dynamic and capable media processing system.

1 Introduction

The *VuSystem* [2, 4] is a programming environment for the development of computer-participative multimedia applications. It is designed to run on high performance computer systems not specifically designed for the manipulation of digital video. The system is unique in that it combines the programming techniques of visualization systems and the temporal sensitivity of traditional computer-mediated multimedia systems.

VuSystem code is split into two partitions: one which does traditional *out-of-band* processing and one which does *in-band* processing. Out-of-band processing is that processing which performs event handling and other higher-order functions of a program. In-band processing is the processing performed on every video frame and audio fragment.

This architecture differs from that of traditional multimedia toolkits that either provide a low level language interface to a library of primitives [12, 13], or provide a high level language system with limited extensibility [11]. In the VuSystem, in-band code can be written in a low level language and can be optimized for performance, while out-of-band code can be written in a high level language and optimized for programmability, usability and extensibility.

In the VuSystem, the in-band processing partition is arranged into processing *modules* which logically pass dynamically-typed data *payloads* though input and output *ports*. These in-band modules can be classified by the number of input and output ports they possess. The most common module classifications are sources, with no input ports and one output port; sinks, with one input port and no output ports; and filters with one input port and one output port. In-band media-processing code is more elaborate and extensible in the VuSystem than in traditional computer-mediated multimedia systems because VuSystem applications perform more analysis of their input media data.

*This research was supported by the Advanced Research Projects Agency of the Department of Defense, monitored by the United States Air Force (AFSC, Rome Laboratory) under contract No. F30602-92-C-0019, and by a grant from Nynex.

†The author can be reached at: MIT Laboratory for Computer Science, Room 504, 545 Technology Square, Cambridge, MA 02139; Tel: +1 617 253 6042; Email: cjl@lcs.mit.edu.

Figure 1: The Sports Highlight Browser, written with the VuSystem. The smaller panes on the right contain scoreboard graphics reporting final scores of sporting events. Clicking on one shows the video highlights of the event in the larger pane.

The nature of out-of-band processing is very different from in-band processing. For the out-of-band code, a programming system can be chosen that can handle user interfaces and other event-driven program functions well. When designing the out-of-band partition, programmability is more important than performance. For maximum ease of application development, the out-of-band partition of VuSystem is programmed in an interpreted scripting language. Application code written in this scripting language is responsible for creating and controlling the network of in-band media-processing modules, and controlling the graphical user-interface of the application.

The scripting language used in the VuSystem is the Tool Command Language, or Tcl [8]. Application code written in Tcl is responsible for creating and controlling the network of in-band media-processing modules, and controlling the graphical user-interface of the application. In-band modules are manipulated with *object commands*, and in-band events are handled with *callbacks*.

The VuSystem is implemented on Unix workstations as a shell-like program that interprets an extended version of Tcl. In-band modules are implemented as C++ classes and are linked into the shell. Simple applications that use the default set of in-band modules are written as Tcl scripts. More complicated applications require linking in additional in-band modules to the default set.

Several computer-participative multimedia applications have been developed with the VuSystem [3]. *The Room Monitor* and *The Sports Highlight Browser* are two representative examples. The Room Monitor processes video from a stationary camera to determine if a room is occupied or empty, and records only when activity is detected above some threshold, producing a series of video clips that summarize the activity in the room. The Sports Highlight Browser (Figure 1) provides access to a sporting news telecast that has been automatically segmented into a set of video clips, each of which represents highlights of a particular sporting event.

VuSystem programs have a *media-flow* architecture: code that directly processes temporally sensitive data is divided into processing *modules* arranged in data processing *pipelines*. This architecture is similar to that of some visualization systems [16, 18], but is unique in that all data is held in dynamically-typed time-stamped *payloads*, and programs can be reconfigured while they run. Timestamps allow for media synchronization. Dynamic typing and reconfiguration allows programs to change their behavior based on the data being fed to them.

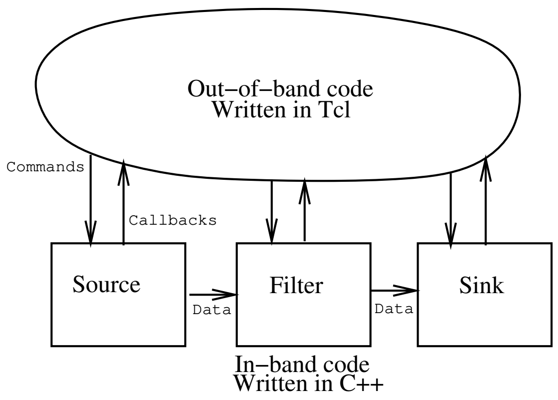

Figure 2: The structure of VuSystem applications. Out-of-band code written in Tcl creates an controls in-band media processing modules written in C++.

2 The Tool Command Language

The Tool Command Language is an excellent programming language for assembling modules into flexible applications. Tcl is designed as a simple but extensible command language. Its syntax is concise enough that simple Tcl commands can just be typed in, but it is programmable and powerful enough that most of the control logic of a large application can be written in it. It has a simple and efficient interpreter, and a simple interface to C.

Tcl syntax is similar to that of Unix shells, but it has additional Lisp-like constructs: Tcl uses curly braces to group elements, square brackets to invoke command substitution, and dollar signs to invoke variable substitution.

Tcl's Simple Type System

In Tcl, all commands and values are strings. There is no other data type. It has no native representation of numbers or lists. All data is in the form of character strings. Even Tcl commands themselves are strings. Since all data in the Tcl interpreter are strings, the embedding interface is simplified. It is easy to pass data between the Tcl interpreter and C code in an application. The C code need only be able to convert internal objects to and from strings. No library of converters between multiple representations is required.

Data types that can be easily represented in string form are quite natural to use within Tcl. For example, numbers can be easily converted to and from strings using standard mechanisms, and lists can be easily represented as strings, using curly braces for grouping. Data types too complex to be efficiently converted to and from strings can be represented in Tcl with *handles* or *object commands*.

Representing Complex Objects in Tcl

Some objects of data types too complex to be efficiently converted to and from strings can be represented with string *handles*. Primitive commands that manipulate these objects use standard methods to convert string handles to objects, and from objects to string handles. A good example of objects that are represented by string handles are open files. The standard Tcl library provides commands to open, close, read and write files. These primitives use file *handles*, short strings that can be converted to and from file descriptors.

The most powerful way to represent objects too complex to be efficiently converted to and from strings is through *object commands*. In this approach, for each object of a complicated data type, a unique Tcl Command is registered in the interpreter. Operations on an object are performed by invoking its Tcl command, with the first argument to the command specifying the operation, and

Source	Function
VsSunAudioSource	Interfaces directly to the audio device on Sun workstations.
VsAudioFileSource	Interfaces to an AudioFile [10] server typically running on a Digital Alpha workstation.
VsSunVfcSource	Interfaces to the VideoPix video capture card on Sun workstations.
VsXVideoSource	Interfaces to video display adapters that have video capture capability through the XVideo X extension.
VsVidboardSource	Interfaces to the Vidboard [9], a LAN-based video capture subsystem.
VsFileSource	Interfaces to files stored using the native VuSystem file format.
VsQtimeSource	Interfaces to Quicktime [12] files.
VsMpegSource	Interfaces to MPEG [15] files.
VsCaptionSource	Interfaces to a closed-caption decoder through a serial line.
VsExternalSource	Assembles separate image files into a sequence of video frames.

Table 1: VuSystem sources.

Sink	Function
VsSunAudioSink	Interfaces directly to the audio device on Sun workstations.
VsAudioFileSink	Interfaces to an AudioFile [10] server typically running on a Digital Alpha workstation.
VsWindowSink	Interfaces to video display adapters through the X Window System.
VsFileSink	Interfaces to files stored using the native VuSystem file format.
VsQtimeSink	Interfaces to Quicktime [12] files.
VsExternalSink	Causes a sequence of video frames to be written to several image files.

Table 2: VuSystem sinks.

the rest of the arguments specifying the arguments to the operation. The Tk graphical user interface toolkit uses object commands to manipulate widgets [8].

Object commands are used by the VuSystem to manipulate media processing modules. Each module is manipulated with its own object command. Each object command has several subcommands that allow the state of its object to be queried and changed. VuSystem object commands are constructed using Object Tcl [5], a dynamic object-oriented extension to Tcl that was developed for this purpose.

3 Manipulating Modules

VuSystem media processing modules are created in Tcl with *class* commands and manipulated with *object* commands. A class command is defined for each type of module that can be created, and an object command exists for each module created. For example, the VsWindowSink Tcl command creates a VsWindowSink module, and installs a new command in the Tcl interpreter to control

Filter	Function
VsJpegC/VsJpegD	Compresses/decompresses video frames using the JPEG [14] video compression standard.
VsCCCC/VsCCCD	Compresses/decompresses 8-bit color video frames using a color cell compression algorithm.
VsQRLC/VsQRLD	Compresses/decompresses grayscale video frames using a simple run-length coding scheme.
VsColor24to8/VsColor8to24	Converts between 24-bit color and 8-bit color video frames.
VsColor24toGray	Converts 8-bit color to grayscale video frames.
VsChannelSelect	Only passes payloads with certain channel identifiers.
VsChannelSet	Sets channel identifiers for VsChannelSelect.
VsReTime	Changes the starting-time and duration on payloads.
VsStepper	Allows explicit lockstep scripting control of each video frame passed.
VsWait	Waits for VsFinish payloads.
VsRateMeter	Measures the flow rate of payloads through it.
VvEdge	Performs edge processing on video frames.
VvThresh	Performs thresholding on pixel values.
VvHistogram	Converts a video frame into a pixel value histogram.
VsTcpClient	Interfaces to the client end of a TCP connection.
VsTcpServer	Interfaces to the server end of a TCP connection.

Table 3: VuSystem filters.

Module	Function
VsDup	Copies payloads to provide identical output sequences.
VsTap	Copies payloads to provide identical output sequences.
VsDeMux	Splits its input sequence based on payload channel number.
VsMerge	Merges multiple payload sequences into a single sequence.
VsMux	Merges multiple payload sequences into a single sequence.
VsOrderedMux	Merges multiple payload sequences into a single sequence, but ensures that timestamps in the output sequence are always monotonically increasing.
VsFade	Provides a fade effect between two sequences of video frames.

Table 4: VuSystem modules with more than one input and output.

the module. The VsWindowSink Tcl command takes as its first argument the name of the object command to create. The rest of the arguments to the class command are parameters for the new module.

Module Types

VuSystem modules are best categorized by how many input and output ports they have. A module is either a *source*, a *sink*, a *filter*, or some *other* module.

Modules with no input ports and one output port are called *Sources* because they appear to the VuSystem to source data. Sources typically interface to media capture devices or media storage systems. *Audio* sources interface to audio capture hardware. *Video* sources interface to video capture hardware. *File* sources interface to files. More exotic sources exist as well. Some sources provided with the VuSystem are listed in Table 1.

Modules with one input port and no output ports are called *Sinks* because they appear to the VuSystem to sink data. Sinks typically interface to media playback devices or to media storage

Subcommand	Modules	Function
callback	All Modules	Get and set the callback.
children	All Modules	List all children in the structural hierarchy.
color	Video Sources	Toggle between color and grayscale.
destroy	All Modules	Destroy module and all its children.
frameRate	Video Sources	Get and set the video frame rate to sample.
gain	Audio Sources and Sinks	Get and set the gain.
hue	Video Sources	Get and set the hue.
pathname	File Sources and Sinks	Get and set the pathname.
port	Many Sinks	Get and set the hardware output port.
port	Many Sources	Get and set the hardware input port.
scale	Video Sources	Get and set the video frame size.
start	All Modules	Start module and all its children.
stop	All Modules	Stop module and all its children.

Table 5: VuSystem object subcommands.

Condition	Modules	Indication
caption	VsWindowSink, VsCaptionSink	A caption was received.
compressRatio	Compression Filters	Filter is achieving this compression ratio
detect	VsPayloadDetect	A payload of the specified type has been detected.
done	Effects Modules	The effect has completed.
rate	VsRateMeter	Payloads are passing at this rate.
sinkFinish	All Sinks	A VsFinish payload was received.
sinkStop	All Sinks	A VsFinish payload was received while in stopping mode.
solve	VsPuzzle	The puzzle has been solved.
sourceEnd	All Sources	The end of the source has been reached.
stepDone	VsStepper	A payload has been sent.
waiting	VsWait	A change of waiting status has occurred.

Table 6: VuSystem callback conditions.

systems. *Audio* sinks interface to audio playback hardware. *Video* sinks interface to video playback hardware. *File* sinks interface to files. Some sinks provided with the VuSystem are listed in Table 2.

Modules with one input port and one output port are called *filters* because they are typically used to perform signal processing operations on the data flowing through them. *Compression* filters compress or de-compress video frames. *Pixel format conversion* filters convert the format video frames. *Descriptor* filters perform operations on the descriptors of payloads. *Visual* processing filters perform various functions on video data. Some filters provided with the VuSystem are listed in Table 3.

Modules with more than one input or output ports provide mechanisms for splitting and merging payload sequences. Some modules with one input port and many output ports split a single timestamped sequence of payloads into multiple sequences. Some modules with many input ports and one output port merge multiple sequences into a single sequence. Others combine payload sequences from two input ports to one output payload sequence, operating on the contents of the data. Table 4 lists some VuSystem modules with more than one input and output port.

4 Communication Between In-Band and Out-Of-Band

In the VuSystem, out-of-band Tcl scripts and in-band C++ modules communicate through *object commands* and *callbacks*:

- Out-of-band code is able to create and destroy in-band modules, query the state of in-band modules, and give commands to in-band modules, all through special Tcl *object commands* defined for each in-band module and port.
- In-band media-processing code signals out-of-band Tcl code whenever an appropriate in-band event occurs through Tcl *callbacks*.

Object commands are always completed in the in-band partition *synchronously* with the out-of-band requester: object commands execute immediately and completely when called from out-of-band scripts. In contrast, because of the time-critical nature of in-band code, it is unacceptable for in-band code to wait for a response from out-of-band code. Tcl *callbacks* are executed in the out-of-band partition *asynchronously* with the in-band partition: callbacks are only queued for execution when invoked from in-band code. Later, the VuSystem scheduler actually executes them. Since out-of-band callbacks do not execute immediately when they have been signalled by in-band code, they are only used to signal events to the out-of-band code, and cannot return values to their in-band signallers. Any in-band changes made by an out-of-band callback are performed through object commands.

Object Subcommands

Each module object command has a set of *subcommands* that vary according to type of the module. The internal state of the module can be queried and changed with some of these subcommands. For example, the VsVidboardSource module has a `port` subcommand that is used to control from which input port it captures video. Table 5 shows a list of typical module subcommands.

Subcommands are implemented as normal Tcl command procedures, whose client data argument by default is a pointer to the associated module. Subcommands are declared *friend* procedures to the module class, so they may manipulate private members of a module.

Callbacks

In-band modules process continuous sequences of media data, while out-of-band Tcl control processing deals with events. Sometimes, out-of-band Tcl code in an application should be executed when an in-band event occurs. In this case, an in-band module would call a Tcl *callback*. The VuSystem provides a facility for each module to have a callback, which can be called whenever a specific event occurs during in-band processing. For example, the VsFileSource module calls its callback when it reaches end-of-file.

Callbacks are defined in Tcl. Typically the Tcl application programmer uses a name of a Tcl procedure as the callback command. The Tcl procedure looks at its arguments to determine what event has occurred. Tcl callback commands are installed with the `callback` subcommand. Each module can have only one callback installed at a time. If a module can signal more than one type of event, it supplies keyword arguments to the callback command, so the command can determine which event occurred. Table 6 shows a list of typical module callbacks.

Example

Figure 3 shows how a Tcl application programmer might make use of a simple file source callback that indicates end-of-file. This example code provides the automatic switching of the file source from the file first.uv to the file second.uv when end-of-file is encountered on the first file.

The `sourceCallback` procedure takes a keyword argument list in its args parameter. It extracts any `-sourceEnd` keyword parameter with the `keyarg` command, defaulting to 0. If `sourceEnd` is nonzero, `sourceCallback` changes the file for the source module using the `pathname` subcommand for

```
proc sourceCallback {args} {
  set sourceEnd [keyarg -sourceEnd $args 0]
  if $sourceEnd {
    vs.source pathname "second.uv"
    vs.source callback ""
  }
}

SimpleFileSource vs.source \
  -pathname "first.uv" \
  -callback "sourceCallback"
```

Figure 3: How a simple file source callback might be used.

the module. This will cause the source module to start on the file second.uv. The `sourceCallback` procedure also clears the callback for the source module using the `callback` subcommand for the module so that when the end of second.uv is signalled, it does not run again.

After defining the `sourceCallback` procedure, a SimpleFileSource named `vs.source` is created, with its input file set to first.uv and its callback set to `sourceCallback`. When started, vs.source will read from first.uv and evaluate the Tcl command string "`sourceCallback -sourceEnd 1`" when it encounters end-of-file.

5 The VuSystem Application Shell

The VuSystem is implemented as a Unix *application shell*: it is program that interprets an extended version of Tcl. Linked into the program are all standard in-band modules, implemented as C++ classes. Tcl scripts implement simple applications that use the default set of in-band modules. By linking additional in-band modules into the application shell, more complicated applications can be constructed.

The application shell defines the interface between the primitive module and command developer and the application script developer. At the primitive level, the module developer creates new primitive VuSystem modules and primitive Tcl commands and links them into the VuSystem application shell. At the application level, the developer writes Tcl code that runs in the application shell.

6 Programming the Graphical User Interface

I implemented a Tcl interface to the X Window System Toolkit [17] and the Athena widget set for the graphical user-interface code. At the start of VuSystem development, I chose to use Xt and the Athena widget over the Tk widget set provided with the Tcl distribution, because at that time the Xt intrinsics and Athena widget set were more robust and complete, and also had built-in features for scheduling in large applications. Through the scheduling interface provided by the Xt intrinsics, I provided scheduling to the in-band modules. Today, the VuSystem could be changed to use Tk, since these capabilities are now provided by Tk.

The TclXt and TclXaw components of the VuSystem [4] provide Tcl programming interfaces to the standard X Window System Xt and Xaw libraries. These components enable the Tcl programmer to construct graphical user interfaces based on the Xt toolkit and the Athena widget set. TclXt and TclXaw use object commands to manipulate X displays, application contexts, events, widgets, and widget classes. Widget resources and other object state can be manipulated through subcommands to these object commands. They provide an interface to the Athena widget set that is similar to Tk, as follows:

- Widget instances are created by invoking a WidgetClass command. For example, to create a button, one uses the Command Tcl command, which creates a Command widget.

- Initial values for widget resources are provided as keyword arguments to the class command. The standard string conversion facilities provided by the Xt intrinsics convert the resource values from strings.
- Resources of existing widgets can be queried and changed through widget *sub-commands*. Just as for the initial values of these resources, the standard string conversion facilities provided by the Xt intrinsics convert the resource values to and from strings.
- *Callbacks* and *Translations* can be specified as Tcl commands. These commands are executed whenever the given input event occurs.

TclXt and TclXaw provide a powerful and complete interface to the Athena widget set. They make the entire interface to the Xt and Xaw libraries acessible to the Tcl programmer. With TclXt and TclXaw, there is no library interface available to the C programmer that is not also available to the Tcl programmer.

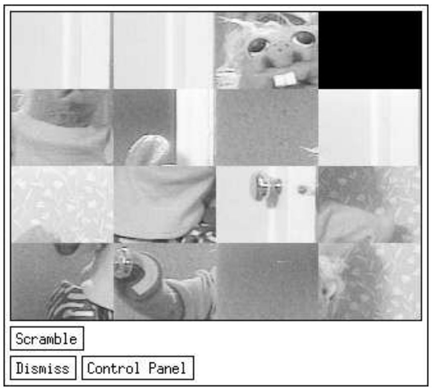

Figure 4: A video puzzle application.

7 Example Application Script

The following is a simple example application built out of a video source module, a video sink module, and a filter module. It implements a video version of a 16-square puzzle. It takes input from a camera or other video source, scrambles it, and then presents the output on a window. Figure 5 shows a block diagram of the in-band modules employed in this application.

The Tcl script fragment that configures the modules and starts them running is shown in Figure 6. This fragment is not complete, but captures the essence of the larger script.

This script first creates an instance of a VidboardSource module and names it `m.source`. Then the script creates an instance of the Puzzle module and names it `m.filter`. It also instructs this module instance to connect its input port to the output port of the `m.source` module instance. The output port of `m.source` was automatically named `m.source.output`.

Next, the script creates a WindowSink module, specifying with the `-window` option that the widget named `w.screen` is the window on the screen to use. The input port of the WindowSink module is also specified to be connected to the output port of the Puzzle module.

Finally the parent module instance named `m`, created before this script fragment was run, is given the `start` command, causing all its child module instances, those with name `m.`*whatever*, to start.

In addition to this script fragment, the puzzle application includes similar code to construct its user interface, and code to cause the `m.puzzle` module instance to reconfigure itself whenever the user moves a block in the puzzle.

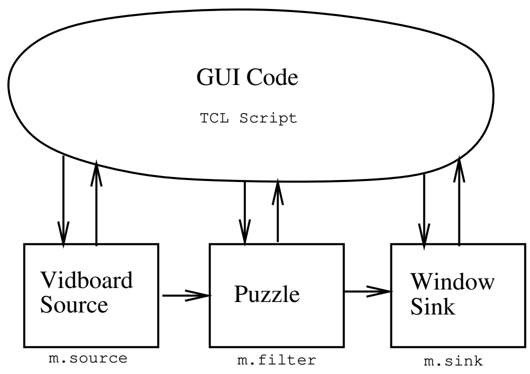

Figure 5: Block diagram of a video puzzle application.

```
VidboardSource m.source

Puzzle m.filter \
  -input "bind m.source.output"

WindowSink m.sink \
  -window w.screen \
  -input "bind m.filter.output"

m start
```

Figure 6: Tcl code from a video puzzle application.

8 Extensions to the VuSystem

A few projects are under way to extend the scope of the VuSystem. Work in progress includes a visual programming system for media computation and a distributed programming system for media applications.

Interactive Programming

A visual programming system for application users is nearing completion. Users interact with the system through a flow graph representation of the running program to control its media processing component. A "flow graph" perspective emphasizes the computation that occurs, rather than a "hypermedia" perspective, which may view the media in terms of a database to be navigated. Flow graph, or dataflow, representations have been used with success in prior visual languages [18].

The visual environment is suited to tasks such as customization, rapid prototyping and experimentation, as well as more general program development. It provides a programming ability (rather than a limited set of configuration options) to users, allowing them to re-program previously developed applications. By embedding it in a toolkit, consistent user programming facilities are available in all derived applications.

From the user's point of view, the visual environment consists of a number of display windows (Figure 7). One window shows the media flow graph representation, the primary means of user programming. Another is a customization panel, detailing the individual options that may be selected for each module of the program's computation. Further objects allow interactions with the textual programming methods of the VuSystem. Description panels show the code fragment associated with a module, and an Interpreter window evaluates commands on demand.

The visual environment demonstrates the modularity, flexibility, and extensibility of Tcl and the VuSystem. Its implementation required only a few additions to the VuSystem core; it works from within the VuSystem; and it is mostly written in Tcl. The visual environment also demonstrates

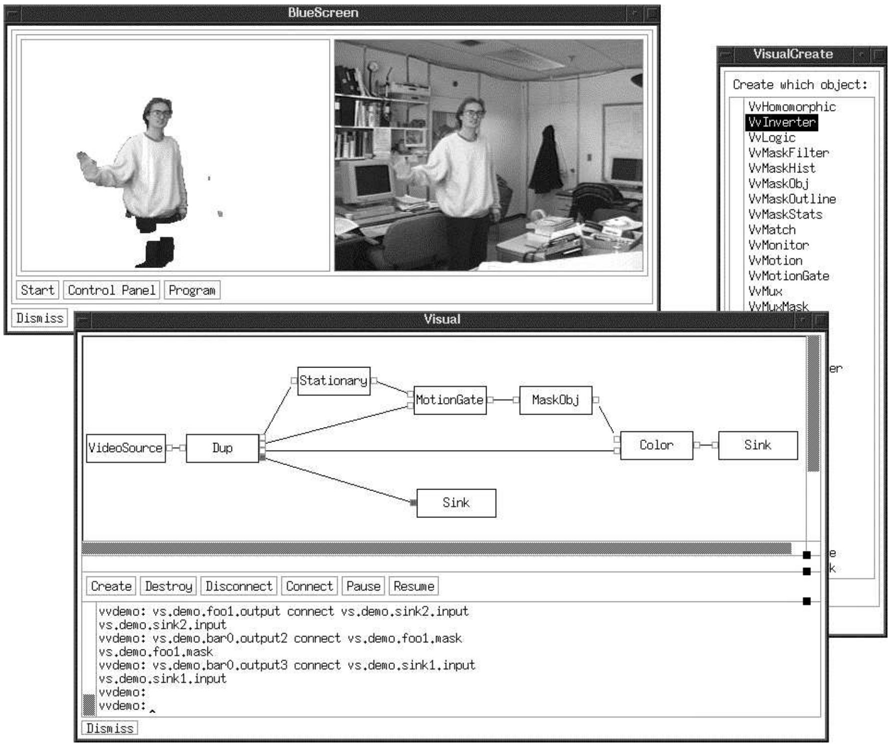

Figure 7: The visual programming interface to the VuSystem, manipulating some vision service modules. The program running is a smart blue-screen application, which separates the active foreground from the stationary backgound in a video sequence. The diagram is of the VuSystem modules that comprise the program.

the introspective capabilities of Tcl and of the VuSystem. The environment shows that VuSystem applications can examine and modify themselves while they run.

Distributed Programming with VuDP

Currently, all modules of a standard VuSystem program must execute in the same local environment on a single host. Applications split across the network must be realized as a set of co-operating programs, making them difficult to write. Using VuDP, a program may be constructed from modules that exist in different environments distributed across several hosts.

Under development, the VuSystem Distributed Programming (VuDP) extension will simplify the construction of applications whose processing is distributed across the network. Three advantages offered by the VuDP model are: transparent access to shared resources, the ability to divide applications across hosts, and the enabling of collaborative applications.

VuDP will support distributed programming through three mechanisms: a remote evaluation capability, an extensible set of exportable services, and a network based model for intra-application communication. The remote evaluation capability will provide a general means of creating remote modules in a suitable execution environment. The exportable services will work in an RPC-like manner to support common tasks. Finally, VuDP will support both in-band communication for media flow and out-of-band communication for control between modules at different sites.

VuDP will leverage off features of Tcl and the VuSystem that make distributed systems easy to implement. The remote evaluation component of VuDP will work by passing Tcl commands and values over reliable byte-stream TCP connections. Since all commands and values in Tcl are strings, linearization for this network transport is trivial. In addition, linearization code for in-band

VuSystem payloads has already been implemented in support of the native VuSystem file format. This support will also be used for the transport of in-band data over reliable byte-stream connections. Distributed VuSystem applications will be built using this mechanism.

9 Conclusion

The VuSystem is a programming environment for computer-participative multimedia applications. It is unique in that it combines the computational flexibility of visualization systems with the temporal sensitivity of multimedia systems. VuSystem applications are partitioned into *in-band* code that manipulates the audio or video data, and *out-of-band* code that performs higher-level event-driven functions.

The in-band partition of the VuSystem is structured as a reconfigurable directed graph of *modules* that logically pass timestamped *payloads* holding media data. The out-of-band partition of VuSystem is programmed in Tcl, an interpreted scripting language well suited to the task. Application code written in Tcl is responsible for creating and controlling the network of in-band media-processing modules, and controlling the graphical user-interface of the application. In-band modules are manipulated with Tcl *object commands*, and in-band events are handled with Tcl *callbacks*.

The VuSystem is designed to be as portable as possible, but includes support for some proprietary media capture and presentation interfaces. The system works best on two specific workstation platforms: The Sun SPARCstation 10/512 and the Digital DEC 3000/400. On the Sun SPARCstation, VideoPix and SunVideo cards are used for video capture, and a standard X server for video output. Sun's audio hardware and software are used for audio input and output. On the Digital DEC 3000/400, DEC's J300 card and the Vidboard [9], a LAN-based video capture subsystem, are used for video capture. The standard X server is used for video display, and DEC's audio hardware and AudioFile [10] software are used for audio input and output.

Tcl is an excellent programming language for out-of-band control in the VuSystem. Its extensibility and simple interface to C is used to a great extent in the VuSystem through object commands. The simplicity of the language and its interpreted nature provide for the rapid prototyping of new VuSystem applications. Its simple type system speeds the implementation of remote-evaluation support for distributed applications. Finally, the introspective features of Tcl and the VuSystem ease the development of interactive visual media programming systems.

References

[1] D. J. Wetherall and C. J. Lindblad, "Extending Tcl for Dynamic Object-Oriented Programming," *submitted to the 1994 USENIX Symposium on Very High Level Languages*, October 1994.

[2] C. J. Lindblad, D. J. Wetherall, D. L. Tennenhouse, "The VuSystem: A Programming System for Visual Processing of Digital Video," *Proceedings of ACM Multimedia 94*, October 1994.

[3] C. J. Lindblad, D. J. Wetherall, W. F. Stasior, J. F. Adam, H. H. Houh, M. Ismert, D. R. Bacher, B. M. Phillips, D. L. Tennenhouse, "ViewStation Applications: Intelligent Video Processing Over a Broadband Local Area Network," *Proceedings of the 1994 USENIX Symposium on High Speed Networking*, August 1994.

[4] C. J. Lindblad, "A Programming System for the Dynamic Manipulation of Temporally Sensitive Data," MIT/LCS/TR-637, MIT Laboratory for Computer Science, Cambridge, MA, August 1994.

[5] D. J. Wetherall, "An Interactive Programming System for Media Computation," SM Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, September 1994.

[6] D. R. Bacher, "Content-Based Indexing of Captioned Video," SB Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA, May 1994.

[7] D. L. Tennenhouse, J. Adam, D. Carver, H. Houh, M. Ismert, C. Lindblad, W. Stasior, D. Wetherall, D. Bacher, and T. Chang, "A Software-Oriented Approach to the Design of Media Processing Environments," *Proceedings of the International Conference on Multimedia Computing and Systems*, May 1994.

[8] J. K. Ousterhout, "Tcl and the Tk Toolkit," Addison Wesley, 1994.

[9] J. F. Adam, "The Vidboard: A Video Capture and Processing Peripheral for a Distributed Multimedia System," *Proceedings of the ACM Multimedia Conference*, August 1993.

[10] T. M. Levergood, A. C. Payne, J. Gettys, G. W. Treese, and L. C. Stewart, "AudioFile: A Network-Transparent System for Distributed Audio Applications," *Proceedings of the USENIX Summer Conference*, June 1993.

[11] Apple Computer Inc., "Hypercard (Version 2.2)," Apple Computer Inc., 1993.

[12] Apple Computer Inc., "Inside Macintosh: Quicktime, Inside Macintosh: Quicktime Components," Addison Wesley, 1993.

[13] Microsoft Corporation, "Microsoft Video For Windows Users Guide," 1992.

[14] ISO/IEC JTC1/SC2/W10, "Digital Compression and Coding of Continuous-Tone Still Images," IEC Draft International Standard 10918-1, 1992.

[15] ISO/IEC JTC1/SC29, "Coded Representation of Picture, Audio, and Multimedia/Hypermedia Information," Committee Draft of Standard ISO/IEC 11172, 1991.

[16] C. Williams and J. Rasure, "A Visual Language For Image Processing," *IEEE Computer Society Workshop on Visual Languages*, Skokie, Illinois, 1990.

[17] P. J. Asenta and R. R. Swick, "X Window System Toolkit: The Complete Programmer's Guide and Specification," Digital Press, 1990.

[18] C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. van Dam, "The Application Visualization System: A Computational Environment for Scientific Visualization," *IEEE Computer Graphics and Applications*, pp. 30-42, July 1989