

# The Mash Programming Environment

*Lawrence A. Rowe*

University of California, Berkeley

URL: *<http://www.BMRC.Berkeley.EDU/~larry>*

# What is Mash?

- **Multicast streaming media toolkit**
  - A public domain, portable research toolkit
  - Mbone community developed conferencing tools (e.g., *sdr*, *vic*, *vat*, *nv*, *wb*, *rat*, etc.)
  - McCanne developed Mash starting with *vic/vat* code
- **Mash abstractions/features**
  - Devices, codecs, networks/protocols, buffers, agents, applications, announce/listen managers, etc.
  - Applications including *collaborator*, *vic/vat*, *nsdr*, *mb*, *dc*, etc.

# Mash Implementation

- Uses split-system object-oriented architecture
  - High performance routines coded in C++
  - User interface and low performance routines coded in Tcl/Tk/Otcl scripting language
  - Tclcl binds C++ and Otcl classes and methods
- 425,000 lines of code
  - 290,000 lines of C++
  - 135,000 lines of Tcl
- Supported platforms
  - Unix (Linux, FreeBSD, Sun Solaris)
  - Windows (WNT, W98)

# Open Mash Consortium

- Research community needs an open source portable toolkit for building streaming media applications
  - Want to build/experiment with applications not write yet-another streaming media toolkit
- Requirements
  - Supported on major platforms (e.g., Unix, Windows, and Macintosh)
  - Documentation and tutorials
  - Someone needs to provide support and act on behalf of the community

# Open Mash Consortium (cont.)

- Why Mash?
  - Large Mbone community
  - Mash has considerable functionality already (e.g., media support, protocols, abstractions, applications, etc.)
- But isn't Java...
  - Sure, but run-time performance really matters (e.g., software codecs, garbage collection?, etc.)
- But Tcl/C++/TclCL is a mess...
  - Yes it is, but we are making lemonade!
  - Modelled after BSD project and other successful Open Source initiatives

# Open Mash Consortium (cont.)

- NSF has provided 3 years of funding
  - Hire two full-time programmers and several part-time people to work on website, documentation, and multimedia content
  - Run workshops, supervise advisory board, and support the community
  - Support several graduate students to work on various aspects of the software
- Need to hire programmers but we don't offer stock options!

# Information

- Website <http://www.openmash.org/>  
Source code, newsgroups, papers, applications, etc.  
Public and developer CVS archive
- Newsgroups  
[openmash-list@openmash.org](mailto:openmash-list@openmash.org)  
[openmash-announcements@openmash.org](mailto:openmash-announcements@openmash.org)  
[openmash-list-request@openmash.org](mailto:openmash-list-request@openmash.org)
- Papers, presentations, ...  
Many research papers, but limited documentation  
These webcasts will be published (slides+video) for on-demand viewing

# Outline

- Tcl/Tk Introduction
- Object-oriented Tcl (Otccl)
- Mash Abstractions
- Simple Mash Applications

# Tcl/Tk Introduction

- Designed as interpreted language that could be added to your app to improve flexibility
  - Modelled on *cs* language (ugh!)
- Variety of data and control abstractions
  - Values are strings
  - List and associative array data structures
  - Assignment, conditional, and looping statements
- Usual procedural abstractions
  - Tcl procedures with local/global variables
  - By value parameter passing with named and default parameters
  - Can add new commands as Tcl or C/C++ procedures

# Evaluation Rules

```
command arg0 arg1 ... argn
```

- Evaluate arguments and apply command
- `$x` returns value of variable `x`
- `[...]` evaluates enclosed Tcl command
- `{...}` substitute values for variables/commands
- `"..."` do not substitute values for variables/commands

# Simple Examples

```
% set x "Hello World"           # define a variable
Hello World
% puts $x                       # print the value of the variable
Hello World
% set len [string length $x]    # assign result of command evaluation
10
% puts "The length of $x is [string length $x]" # quotes do eval
The length of Hello World is 10
% puts {The length of $x is [string length $x]} # braces do not eval
The length of $x is [string length $x]
% set x                         # setting variable returns value
Hello World
```

# Calling a Procedure

## *Define Procedure*

```
proc getopt { addr bw {128000} } {  
    if {$addr == "public"} {  
        if {$bw > 128000} { set bw 128000}  
    }  
    return $bw  
}
```

## *Call Procedure*

```
% getopt public  
128000  
% getopt public 1500000  
128000  
% getopt i2 1500000  
1500000
```

# Tcl Commands for Interfaces (Tk)

- Top-level default window named “.”  
Subwindows typically named .x or .x.y.z
- Creating a widget defines a command with the same name  
`button .b ...` defines command “.b”  
`.b configure -color red -text “Hi Folks”`
- Many predefined widgets  
Buttons, menus, listboxes, canvases, images, etc.
- Event dispatching loop implicitly provided  
Can dispatch input event or set your own time events

# Create A User Interface



*Tk Commands to Create “hello” Button Application*

```
% button .b -text "Hello World" -command {puts stdout "hi"}  
.b  
% pack .b          # pack the widget and map window  
% hi  
hi  
hi
```

# Discussion

- Tcl model uses Tcl to combine primitives coded in C/C++
- Tcl applications
  - tclsh* – command shell supports Tcl commands only
  - wish* – command shell that supports Tcl and Tk commands
  - yourapp* – custom defined application with embedded Tcl interpreter
- Dynamic loading
  - Source Tcl scripts or load C/C++ object files
  - C code can define new commands
- Extensions for RPC
  - Uses text-based call protocol which is good for scripting
  - Supports blocking/nonblocking RPC with binary data

# Implementing a Tcl Command in C

```
/* RandomCmd – return random integer. */
int RandomCmd(ClientData clientData, Tcl_Interp *interp, int argc, char *argv[]) {
    int rand, error;
    int limit = 0;
    if (argc > 2) {
        interp->result = "Usage: random ?range?";
        return TCL_ERROR;
    }
    if (argc == 2) {
        error = Tcl_GetInt(interp, argv[1], &limit);
        if (error != TCL_OK) {
            return error;
        }
    }
    rand = random();
    if (limit != 0) {
        rand = rand % limit;
    }
    sprintf(interp->result, "%d", rand);
    return TCL_OK;
}
```

# Loading/Calling *random*

## *Register Command*

```
TclCreateCommand(interp, "random", RandomCmd,  
                (ClientData)NULL, (Tcl_CmdDeleteProc *)NULL);
```

## *Example Use of Command*

```
% load /usr/local/lib/random.so Random  
% random  
172  
% random 475  
13
```

# Object-oriented Tcl

- Object-oriented programming extension to Tcl modelled on Common Lisp Object System
  - Multiple inheritance
  - Dynamic class and method definition
  - Can add variable to any instance at any time
  - Can define procedure on specific instances
- Coded in C but makes extensive use of Tcl implementation internals

# Example

## *Class Definition*

```
Class Stack
Stack instproc init {args} {
    $self next
    $self set things_ $args
}
Stack instproc put {x} {
    $self instvar things_
    set things_ [concat [list $x] $things_]
    return $x
}
Stack instproc get {} {
    $self instvar things_
    set x [lindex $things_ 0]
    set things_ [lrange $things_ 1 end]
    return $x
}
```

## *Usage*

```
% Stack mystack
mystack
% mystack put toast
toast
% mystack put jelly
jelly
% mystack get
jelly
% mystack put jam
jam
% mystack set things_
toast jam
% set another [new Stack coffee]
% $another set things_
coffee
```

# Example (cont.)

*Query info about classes/instances*

% mystack info class

**Stack**

% Stack info instances

**mystack**

% Stack info instcommands

**init put get**

% mystack info vars

**things\_**

% Stack info superclass

**Object**

# Discussion

- Can define class/methods in C++ or Otcl and call them from anywhere
  - Somewhat complicated to call Tcl commands from C++ and to call C++ methods from Tcl, but possible
- Incremental development and prototyping
  - Define class/methods in Otcl
  - Move performance sensitive methods to C++
- Can bind Tcl variables to Otcl objects that represent interface widgets
  - tkvar* and *tkvarname* identify variables
  - Similar to *local* and *instvar* commands

# Mash Abstractions

- **Core Abstractions – required entities**  
Devices, encoders/decoders, framers/fragmenters, buffers, networks, sessions, modules, applications, etc.
- **Library Abstractions – optional services**  
srm, snap, coordbus, cues, scuba, etc.
- **Application Abstractions – user applications**  
sdr, vic, vat, collaborator, mb, mars, dc, psvp, newscast, rtpgw, camera-client/-server, rvic, room-controller, etc.

*Where to begin?*

# A Simple Mash Script

- Connect to SAP channel and call procedure when an announcement is received
- Mash script – not application
- Uses built-in *Announce/Listen* abstraction
  - AnnounceListenManager* – sends/receives announcements as required
  - AnnounceListenManager/SAP* – subclass that knows about SAP announcements

# sap-listen.mash

```
Import enable                # enable class imports
import AnnounceListenManager # both ALM and ALM/SAP

# override default behavior – just print the announcement
AnnounceListenManager/SAP instproc recv_announcement { addr port data size } {
    puts $data
}

# create instance of ALMgr/SAP and start listening
set sdr [new AnnounceListenManager/SAP 224.2.127.254/9875 2048]
vwait forever
```

# Example of Running It

```
csh: setenv TCLCL_IMPORT_DIRS "/home/larry/mash/mash-1/"
csh: mash
% source sap-listen.mash
%
v=0
o=- 961460411 3173027604 IN IP4 171.64.20.134
s=Pine-Paul-Livecast
b=AS:320
t=3173027604 3173029224
a=type:broadcast
a=tool:liveCaster v1.0a51
m=audio 50494 RTP/AVP 14
c=IN IP4 237.112.88.5/15

v=0
o=nrk 3160301525 3160301675 IN IP4 nrkp1.mbone.hiof.no
s=NRK P1 Radio (Høgskolen i Østfold)
i=Sending av NRK P1 Radio fra Høgskolen i Østfold. Dette prosjektet er et samarbeid mellom Høgskolen i
  Østfold og UNINETT.
u=http://www.hiof.no/smm/nrk_pa_net/
e=NRK Alltid Nyheter (Høgskolen i Østfold) <thomas.f.malt@hiof.no>
p=NRK Alltid Nyheter (Høgskolen i Østfold) (+47) 6921 5346
t=0 0
a=tool:thomas.f.malt@hiof.no
a=type:broadcast
...
```

# Announce/Listen Abstraction

- Handles simple text announcements
- User defines...
  - Announcement text
  - Time interval between announcements
  - Callback procedure for received announcements
- Can manage several announcements with different time intervals
- AnnounceListen/SAP has methods to construct and parse session announcements

# Working with Mash Code

- Extremely confusing when you start
  - Gets better when you understand model and implementation designs
  - Need to learn how to find things – poor documentation and ***no class browser*** (yet)
- I will identify where to look for definitions
  - Assumes you are at `.../mash-1/` directory

*net/announce-listen.cc – defines base classes*

*tcl/net/al.tcl – defines programmer abstraction for using the classes*

# Mash Applications

- Application provides mechanisms to...
  - Lookup configuration options (Xdefaults, .mash/prefs, etc.)
  - Parse command line arguments
  - Format/print error messages
- Each application is an instance of a subclass
  - Runs in a process
  - May have GUI (*mash*) or command line interface (*smash*)

# Simple Application

```
Import enable
import Application

Class SimpleApp -superclass Application

SimpleApp instproc init {argv} {
    $self next SimpleApp
    # register command line arguments and setup defaults
    set o [$self options]
    $o register_option -a sessionAddress
    ...
    $o add_default sapZones 224.2.128.0/17,239.255.0.0/16
    $o add_default sapTTL 127
    ...
    # parse command line arguments
    $o parse_args $argv
    # initialize the app - build UI, setup sessions, etc.
    $self init_local
    # call any user-specific code associated with app
    $self user_hook
}

# create app and enter event loop
set app [new SimpleApp]
vwait forever
```

# Example of Running It

```
csh: SimpleApp -sessionAddress 234.2.3.4/9822
```

- Must modify *SimpleApp* file to restart with Mash

```
#!/bin/sh
# the exec restarts using mash which in turn ignores
# the command because of this backslash: \
exec mash-5.0b2 "$0" -name SimpleApp -- "$@"
```

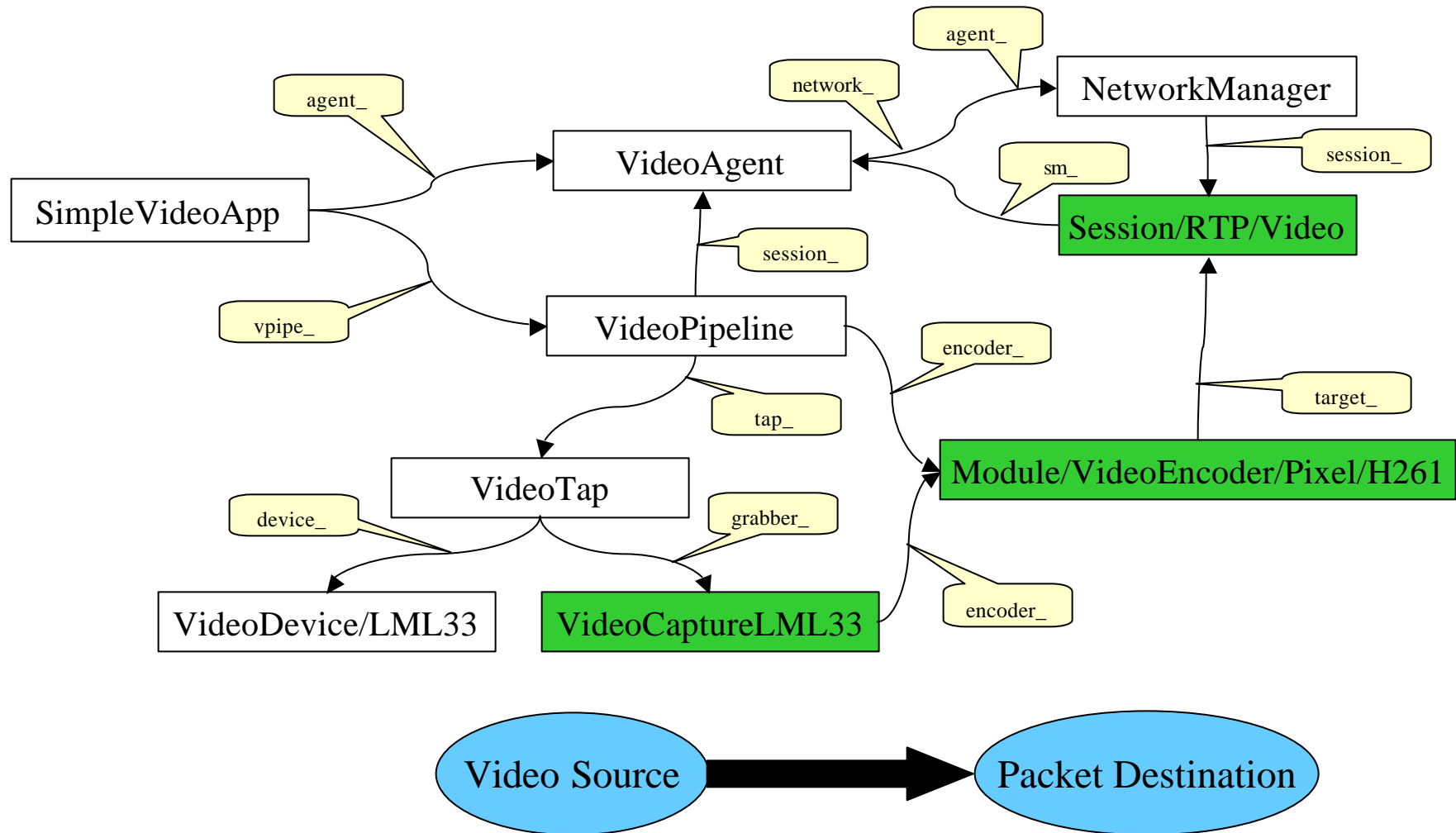
*tcl/common/application.tcl* – defines *Application* class

see following example apps: *tcl/vic/*, *tcl/vat/*, *tcl/nsdr/*, *tcl/applications/newscast/*, etc.

# Let's Stream Media

- Simple application to open device and stream media
- Abstractions used...
  - Device* – Mash abstraction for media devices
  - Agent (VideoAgent, AudioAgent)* – controls media sources and destinations
  - VideoPipeline* – abstracts stream of video
  - VideoTap* – video source (e.g., device, storage, etc.)
- Abstractions are not completely consistent!
  - Media source from multicast session not represented by *VideoTap* and audio devices not treated as subclasses of *Device*

# Overview of Objects



# SimpleVideoApp Code

```
Import enable
import Application AddressBlock VideoAgent VideoPipeline
Class SimpleVideoApp -superclass Application
SimpleVideoApp instproc init {} {
    $self next SimpleVideoApp
    $self add_default defaultTTL 16
    $self instvar agent_ vpipe_
    set agent_ [new VideoAgent $self 224.2.3.4/9822]
    set vpipe_ [new VideoPipeline $agent_]
}
SimpleVideoApp instproc run {} {
    $self instvar vpipe_
    set d [lindex [$vpipe_ input_devices] 1]
    $vpipe_ select $d h261
    $vpipe_ set_quality 10
    $vpipe_ start
}
set app [new SimpleVideoApp]
$app run
vwait forever          #after 3000 [[$app set vpipe_] stop]
```

# Audio Works the Same

- Agents and devices

*AudioAgent* and *Audio* device

- Network abstractions

*NetworkManager* and *Session/RTP/Audio*

- CODEC abstractions

*Module/AudioDecoder, Module/AudioDecoder/ADPCM, .../PCM, .../GSM, .../LPC*

*Module/AudioEncoder, Module/AudioEncoder/ADPCM, .../PCM .../GSM, and .../LPC*

- Buffers, etc.

# Receiving Media

- Receive data and demux
  - Look for decoder for source (i.e., srcid, src ip addr)
  - Create one if new member
- Hand packet to decoder
  - Decode data
  - Prepare it (e.g., mix or color space convert)
  - Video frames may be displayed in different windows at different size/resolution
- Play it
  - AudioController* plays audio blocks
  - Renderer* plays video frames

# Discussion

- Mash has *Observer/Observable* abstractions
  - Observer* instances can be notified when specific *Observable* instances change
- Audio/video player
  - Need *AudioAgent* and *VideoAgent* and two RTP sessions
  - Information needed to create these objects contained in a SAP announcement
- UI coded in Tk
  - Some actions cause changes to Otcl objects and some cause changes to underlying C++ objects
- Remember it is an event-driven system
  - Events: UI events, I/O events, or timer events

# Application Packaging

- Most apps composed of many files
  - vic* is 7,000 lines of Tcl spread over 19 files
  - Mash builds a single script with all that code
  - Running an application starts Mash executable and hands it the script (described earlier)
  - Look at the last lines in the file – usually does “`vwait forever`”
- How would you experiment with the code?
  - Add `puts` commands into code, remove `vwait`, and run from console
  - Or modify source files and rebuild script file

# What Next?

- **Run applications**
  - Download Mash binary distribution which has compiled mash executable and application script files
- **Build an application**
  - Usually just requires that you write Tcl code
  - Find good model (e.g., *nsdr*, *archive/player*, *vic*, etc.) and mimic it
- **Do some serious hacking**
  - Download Mash source distribution which has everything
  - Learn the TclCL abstractions
  - Find another file/abstraction similar to what you want to do and modify it
  - Note: there's a lot of cruft code in the Mash tree

# Summary

Try it – you'll like it!