Results

- Using command languages results in dramatic reduction in syntax size and complexity.
  - 7,000 lines C reduced to 700 lines tcl in tclMotif.
  - Similar results for other systems.
- Fast prototyping via interpreter.

Language issues

- Xt restrains many possibilities to syntax only
  - `.w setValues -value on` (tclMotif)
  - `setValues $w value on` (Wafe)
  - `XtSetValues $w value:on` (WKSH)
- Tk uses “configure” to do XtSet/Get Values
  - `.w confgure -value on`
- Motif requires multiple resource setting, forcing a difference to Tk.
- There is a choice in paradigms
  - action+object (Wafe, WKSH)
  - object+action (tclMotif, Tk)

tclMotif getting values

To interpret

```
.btn getValues -labelString l
```

- Locate resource list for widget.
- Find `labelString` entry.
- Look at converted type, *X.*
- Execute *X* to String converter.
- Set value in Tcl variable `l`.

tclMotif Setting Values

To interpret

```
.btn setValues -labelString "hello"
```

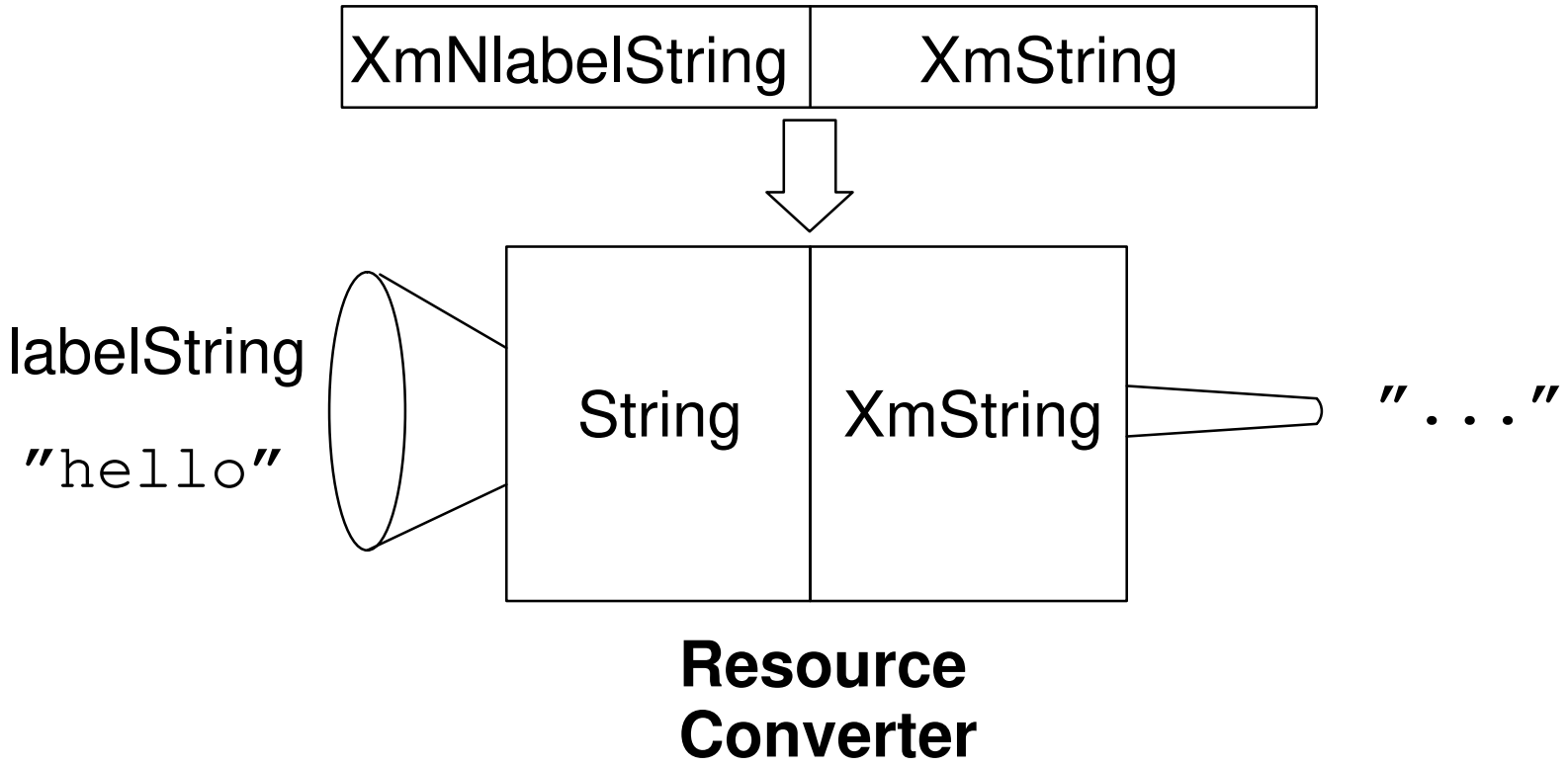
- Locate resource list for widget.
- Find `labelString` entry in this.
- Look at converted type, *X.*
- Call the appropriate String to *X* converter.
- Set the converted value in the widget.

Soft coded resources

- Resources may be set using a *resource file.*

```
.btn.labelString:  hello
```

- Resources are automatically converted by Xt



Xt Resource Mechanism

A resource is any attribute of a widget

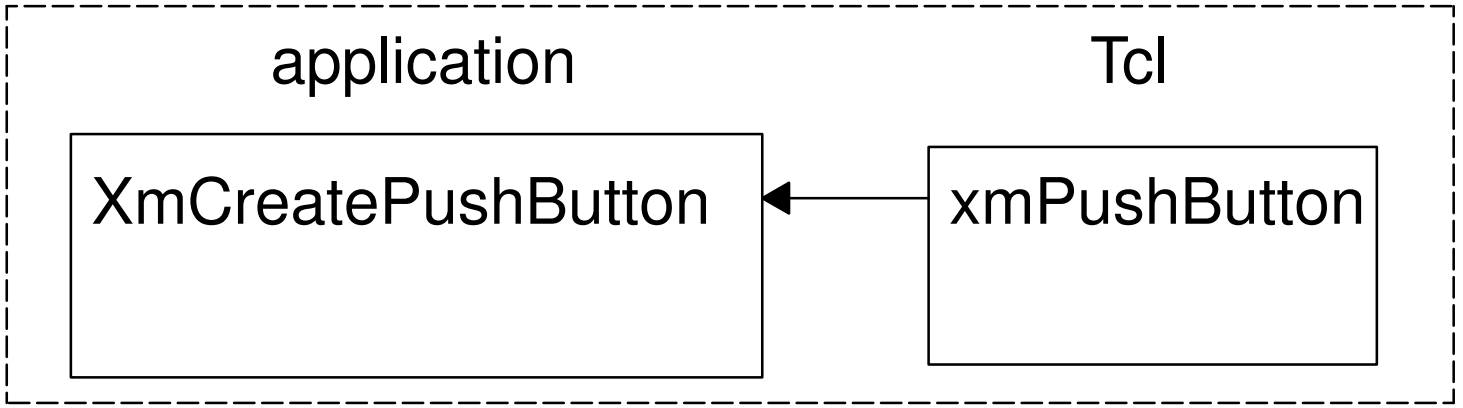
- foreground colour
- width
- label text
- text font
- pushbutton behaviour

Resource specification:

Name	Type	Default
XmNlabelString	XmString	widget name

tcl + Motif

- Define C calls to Motif as tcl commands




```
xmPushButton .btn
```

- Define meaning of arguments to tcl commands

```
.btn setValues -labelString "Hello"
```

Implementation

- Implementation relies on the heavy use of strings in the command language and in the toolkit
  - XmNlabelString = ”labelString”
  - XmARROW_UP = ”arrow_up”
  - Actions defined as strings
  - Translations defined as strings

TkInter summary

- object/action approach
- Complete Tk widget set
- Availability: http://www.cwi.nl/~guido/Python.html

TkInter example

```
from Tkinter import *

class Application(Button):
  def pushed(self):
    print ”I was pushed”

  def __init__(self, master=None):
    Button.__init__(slef, master)
    self.pack()
    self[”text”] = ”Push me”
    self[”command”] = self.pushed

app = Application()
app.mainloop()
```

TkInter

- Python is an O/O language
- TkInter is a binding of the Tk widgets to Python
- TkInter defines an inheritance hierarchy to the (flat) Tk hierarchy

tkPerl status

- In alpha
- Some Tk widgets missing (e.g. Text)
- Shows need to clean up Tk's C interface
- May continue to mirror tcl/Tk or evolve away from tcl dependancies
- Available from ftp://black.ox.ac.uk:/src/ALPHA/tkperl*

tkPerl example

- tkPerl version of earlier Tk example

```
use Tk;
$top = tkinit;
$btn = Button::new($top,
                   ”-text” => ”Push me”);
$btn->configure(”-method” => \&pushedCB);
tkpack $btn;
tkmainloop;
sub pushedCB {
    print ”I was pushed”;
}
```

TkPerl

- Port of the Tk widgets to Perl
- tcl dependencies removed
- Uses Perl5 object mechanisms
- Attempts to optimise out string base of tcl

Vtcl summary

- Simplified interface to Motif and Curses
- Uses action/object approach
- Commercial product
- Limited availability

Vtl example

- Hello world

```
set app [VtOpen Prog1]
set form [VtFormDialog $app.main \
        -ok -okLabel "Goodbye" \
        -okCallback VtClose
set label [VtLabel $form.label \
        -label "Hello world"
VtShow $form
VtMainLoop
```

Xt

- Some name changes
  - XtAppInitialise → VtOpen
  - XtUnmanageChild → VtHide
- Multiple applications can share xm_vserver
- Applications can Lock, Unlock xm_vserver

Widgets

- Motif set plus ComboBox, DrawnList
- Some widgets “modified”
  - Form can have a “reset” button
  - Dialog can have “apply”, “cancel” buttons

Visual Tcl

- Consists of
  - vtcl – per script interpreter
  - xm_vserver – Motif GUI server
  - ch_vserver – Curses server
- Hides Motif and Xt complexity with its own GUI model
- Forms–based GUI model
- Action/object paradigm

Availability

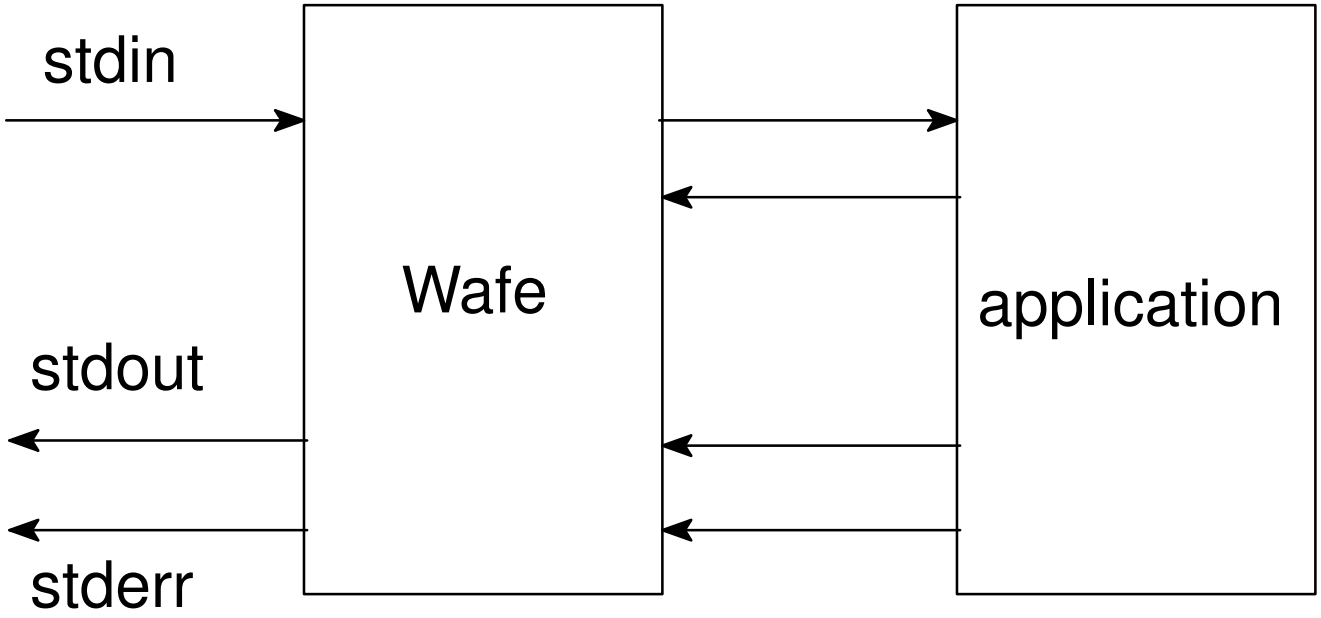
- The authors are Gustaf Neumann, Stefan Nusser
  email: neumann@wu–wien.ac.at
- The primary site is ftp.wu–wien.ac.at:pub/src/X11/wafe
- It is also available from ftp.x.org

Wafe talking to Perl

```
#!/usr/local/bin/perl
$|=1
print
    "%proc pushedCB {} \
            {puts stdout pushed}\n"
    "%mPushButton btn toplevel\n"
    "%callback $btn activateCallback\
             pushedCB\n"

# read loop
while(<STDIN>) {
    # look for "pushed" on stdin
}
```

Wafe frontend structure

- A program separates GUI from application as



- If the application is called “app” then linking wafe to “xapp” sets this up automatically.
- Lines beginning with % sent from the app to Wafe are treated as Wafe commands.

Wafe as frontend

- Primary purpose is as a GUI frontend to applications
- Applications can be in any language with line–buffered I/O
  - Perl
  - Python
  - Prolog

Wafe

- Another tcl to Xt binding
  - Originally to the Athena widgets
  - Later extended to Motif widgets
- Uses action/object model

```
proc pushedCB {} {
     puts stdout pushed
}
mPushButton btn toplevel
callback $btn activateCallback\
       pushedCB
```

tclMotif Availability

- Primary site is csc.canberra.edu.au
- Also on ftp.x.org
- Mailing list via listserv@ise.canberra.edu.au

tclMotif Summary

- Provides interface to full Motif toolkit
- Uses object/action approach
- Publically available source code
- Uses tcl

Other features

- Uses the send mechanism from Tk
- Supports Motif drag and drop
- Support for Motif 2.0 widgets

Other handlers

- Actions handler

```
$w setValues -translations \
   "<Btn3Down>: action(popIt %event)"
```

- Timer handler

```
. addTimer 10000 quit
```

- Input handler

```
. addInput file.txt r readInput
```

```
# add a callback
.main.form.list singleSelectionCallback {
    .main.form.name setValues \
        -value "%item"
}

# set it going

. realizeWidget
. mainLoop
```

```
    -topAttachment attach_form \
    -rightAttachment attach_form \
    -leftAttachment attach_widget \
    -leftWidget .main.form.Name
.main.form.telephone setValues \
    -topAttachment attach_position \
    -topPosition 50 \
    -rightAttachment attach_form \
    -leftAttachment attach_widget \
    -leftWidget .main.form.Telephone

# set values in List
.main.form.list setValues \
    -items {"Jan Newmarch"
            "John Ousterhout"
            "Stephen Prendergast"} \
    -itemCount 3
```

```
.main.form.list setValues \
    -topAttachment attach_form \
    -leftAttachment attach_form \
    -bottomAttachment attach_form
.main.form.Name setValues \
    -topAttachment attach_form \
    -leftAttachment attach_widget \
    -leftWidget .main.form.list \
    -rightAttachment \
          attach_opposite_widget \
    -rightWidget .main.form.Telephone
.main.form.Telephone setValues \
    -topAttachment attach_position \
    -topPosition 50 \
    -leftAttachment attach_widget \
    -leftWidget .main.form.list
.main.form.name setValues \
```

tclMotif Address Book

```
xtAppInitialize

# widgets
xmMainWindow .main managed
xmForm .main.form managed
xmList .main.form.list managed \
    -selectionPolicy single_select
xmLabel .main.form.Name managed \
    -alignment alignment_end
xmLabel .main.form.Telephone managed \
    -alignment alignment_end
xmText .main.form.name managed
xmText .main.form.telephone managed

# geometry
```

tclMotif callbacks

- Arbitrary tcl code as callback function

```
.btn activateCallback {
     puts stdout ”I was pushed”}
```

- Substitutions allow all fields of the “call_data” to be accessed

```
.list singleSelectionCallback {
     listCB %item %item_position}
```

Geometry management

- No change to Motif model

tclMotif functions

- There is a creation function for each widget and convenience function:

```
xmPushButton
xmScrolledList
xmFileSelectionDialog
```

- Each widget has a set of methods
  - Core: setValues, getValues, realizeWidget
  - List: addItem, deleteItem, selectItem
  - Text: copy, cut, findString, setString
  - Toplevel: mainLoop

tclMotif API structure

- A typical program is

```
proc activateCB {...}
xtAppInitialize
xmArrowButton .btn managed
.btn setValues \
     -arrowDirection arrow_up
.btn activateCallback activateCB
. realizeWidget
. mainLoop
```

tclMotif

- `moat` is an interpreter with
  - tcl front–end
  - Xt event loop
  - uses Motif widget library
- Application structure is the same as Xt
- Widget set is the same as Motif
- Creating a widget defines a tcl object, like Tk, with the same naming convention.

Tk availability

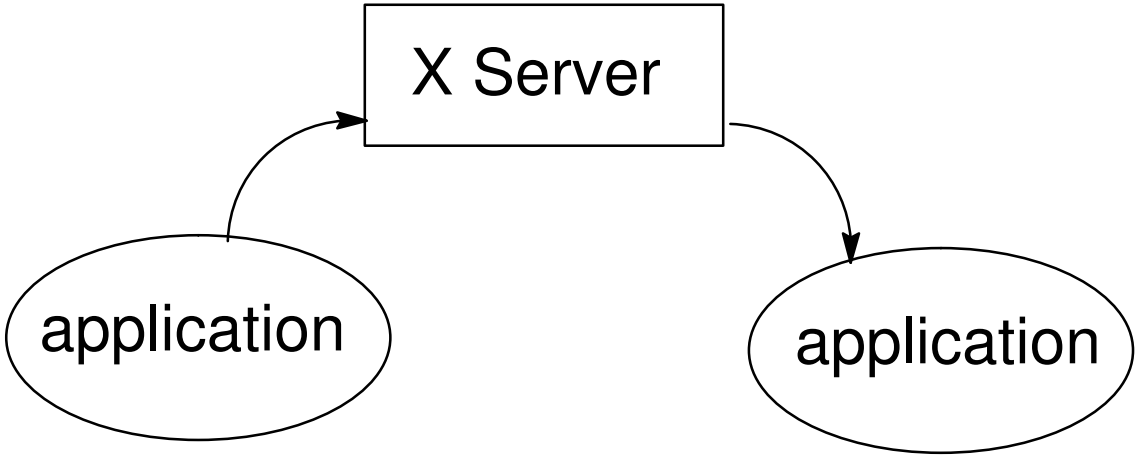
- Author is John Ousterhout
- Main tcl/Tk site is harbor.ecn.purdue.edu
- Also available on ftp.x.org
- Newsgroup is comp.lang.tcl

Tk summary

- Non–Motif toolkit
- Look and feel close to Motif (missing e.g. keyboard traversal)
- Widget set more primitive than Motif
- More 3rd party widgets?
- Uses an open command language, tcl
- Uses an object/action approach
- Publically available source

Send primitive

- Need to decouple GUI from application.
- "send" allows an application to distribute execution to another.




```
send interpreter command
```

- This allows some components to perform GUI functions.
- It allows other components to perform non–GUI functions.

Structured text

- Text allows standard operations
  - edit
  - select
- Locations in text can be marked
- Text tag refers to a range of characters

```
.text tag add x 1.0 1.end
```

  tags line 1 with “x”
- Tagged items can be
  - configured (e.g change font)
  - bound to (e.g. move into text)

Structured graphics

- The canvas widget supports structured graphics:
  Rectangles, circles, ellipses, lines,
  Bezier curves, polygons, arcs,
  text, bitmaps, widgets
- An option to a canvas object is “create”

```
.canvas create rectangle 1c 2c 3c 4c
```

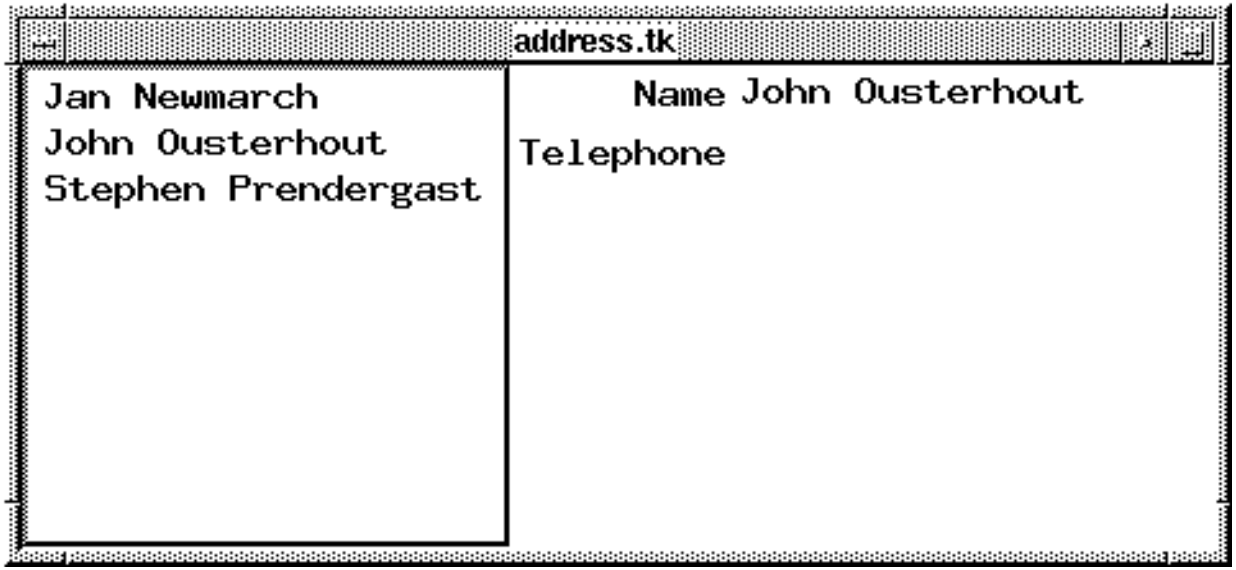
- Items can be tagged to name them
- Tagged items can be
  - moved
  - deleted
  - configured (e.g. change background)
  - bound to (e.g. click on an item)

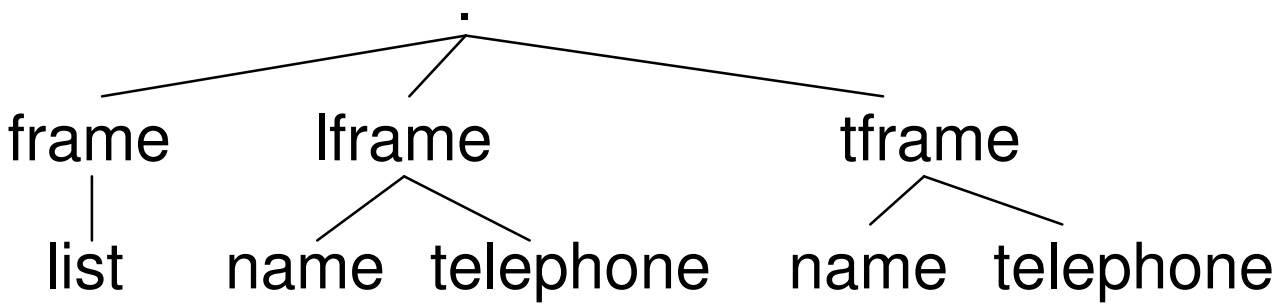
```
pack .lframe.name .lframe.telephone \
    -side top -anchor ne
pack .tframe.name_entry \
     .tframe.telephone_entry \
    -side top -expand 1
pack .frame .lframe .tframe\
    -side left -anchor n
.frame.list insert end \
    {Jan Newmarch}\
    {John Ousterhout} \
    {Stephen Prendergast}
bind .frame.list <Double-Button-1> {
    set index [.frame.list curselection]
    set sel [.frame.list get $index]
    .tframe.name_entry delete 0 end
    .tframe.name_entry insert 0 $sel
}
```

```
#!/usr/local/bin/wish -f
frame .frame \
    -relief raised \
    -borderwidth 2
listbox .frame.list
tk_listboxSingleSelect .frame.list
frame .lframe
label .lframe.name \
    -text Name
label .lframe.telephone \
    -text Telephone
frame .tframe
entry .tframe.name_entry
entry .tframe.telephone_entry
pack .frame.list
```

Address Book in Tk

- The address book again, now in Tk





Bind command

- Some widgets have predefined callbacks

```
button .btn -command pushedCB
```

- To extend this, bind an X event to a widget

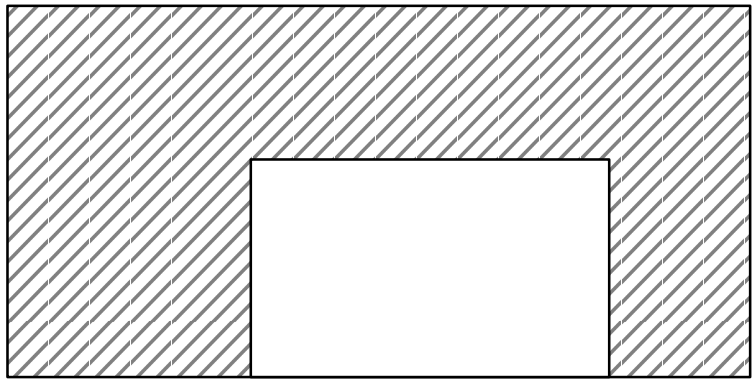
```
bind .list <Double-Button-1> \
     listCB
```

- %substitutions pass call_data into the callback

```
bind .list <Double-Button-1> \
     {listCB %W %x %y}
```

Packer Geometry

- Geometry placement is by filling a decreasing rectangle



- Specify side to fill

```
-side right
```

- Location within space
  - stretch to fill space

```
-fill 1
```

  - anchor to a direction

```
-anchor se
```

Tk widgets

- Primitive widgets include
  Label, Button, CheckButton, RadioButton, Message, Listbox, Scrollbar, Scale, Entry, Text.
- Container widgets include
  Frame, Toplevel, Canvas
- Menus
- Geometry is done by geometry managers
  Packer, Placer

Simple Tk example

- This creates a button which can be pushed

```
proc pushedCB {} {
    puts stdout ”I was pushed”
}
button .btn –text ”Push me”
.btn configure –command pushedCB
pack .btn
```

Wish structure

- General structure of a `wish` application is

```
<Callback function definitions>
<Create and configure widgets>
     <Create widget>
     <Configure widget>
     <Manage widget>
```

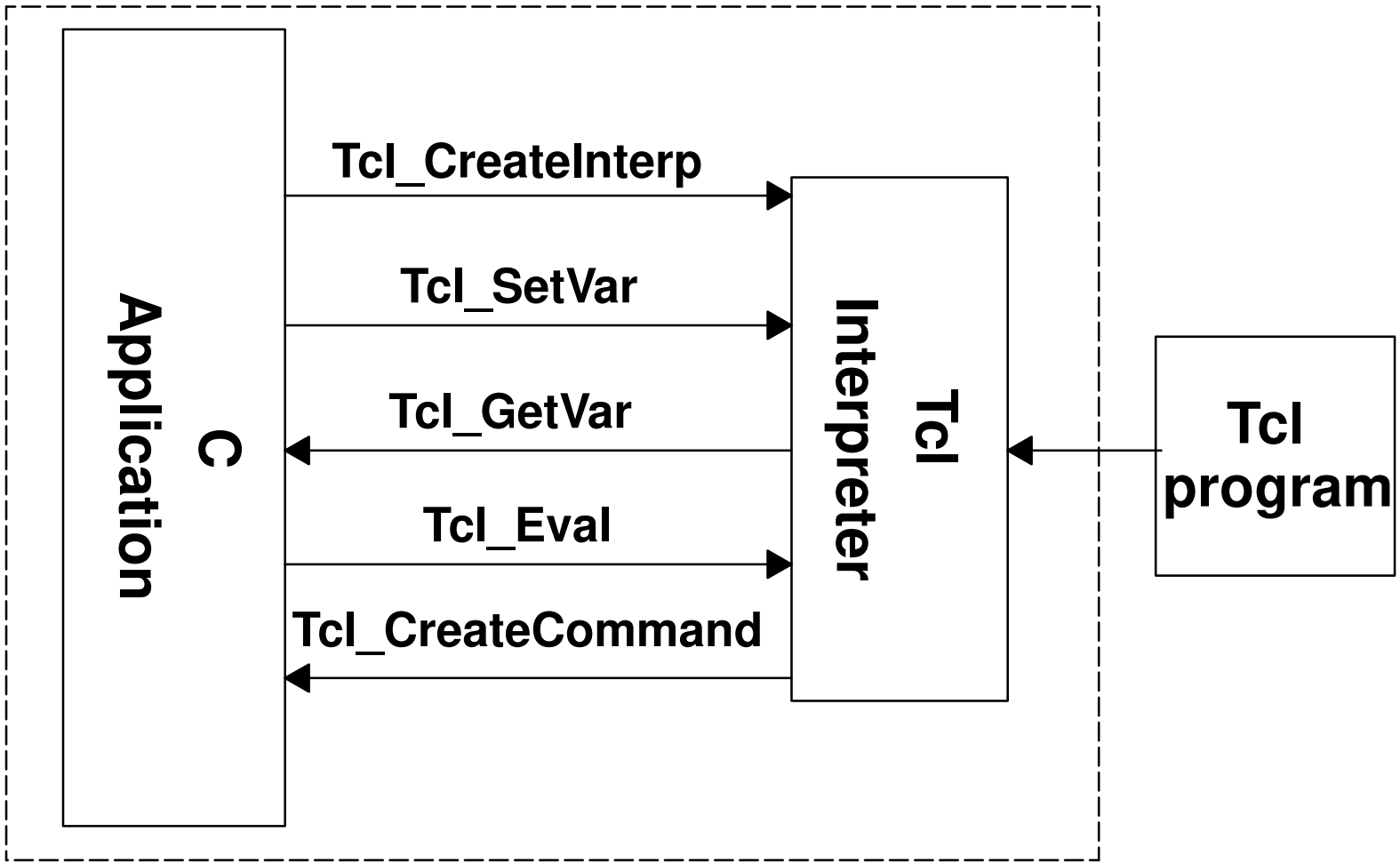
- Tk widgets are new tcl commands (objects).
- The object/widget/command has a name reflecting widget hierarchy

```
.menuBar.file.quit
```

Tk Embedding

- Tk is a library of widgets.
- `wish` is an interpreter with
  - tcl front–end
  - X event loop
  - uses Tk library

Structure of a tcl–enabled application



Sample tcl program

```
# find maximum of a set of numbers
# read from stdin, one number per line

proc max {a b} {
    if {$a > $b} return $a
    return $b
}

gets stdin maximum
while {[gets stdin next] != -1} {
    set maximum [max $maximum $next]
}
puts stdout "maximum value is $maximum"
```

Why use tcl?

- It reduces complexity.
- It allows decoupling.
- It is fun.
- It is free.
- It is supported.
- It is extensible.
- Source code is available.
- It is non–proprietary.
- Saves re–inventing the wheel.

Tcl

- Name is *Tool Command Language.*
- Another command language.
- It is a re–usable tool.
- Independant of any application.
- Can be embedded into *any* application.

WKSH Availability

- Author is J. Stephen Pendergrast, Jr
  email: pend@summit.novell.com
- Acacia Computer,
  PO Box 4376,
  Warren, NJ 07059,
  Phone: 908 548 6955,
  Email: uunet!aca1
- Consensys Corp,
  Europe: +(44)–734–833241 (Roger Chalke),
  +(44)0734–835391 (Fax),
  US: (416)–940–2903, (416)–940–2903 (Fax)
- Computer Aid Inc, 1–(800)–444–WKS

WKSH Summary

- Provides interface to full Motif toolkit.
- Uses a shell that is often claimed to be the next industry standard.
- Uses an action/object approach.
- Commercial product.

WKSH extensions

- The DTKSH (Desk Top KSH) is the WKSH extended to the CDE environment. This offers
  - Extra widgets in CDE
  - Toolktalk interface.
- Object modules in C can be dynamically linked.
- A set of aliases is there to make things suitably compact cmw, crtarrowb, ato, acs, ...

Miscellaneous handlers

- You can set a timer using

```
XtAddTimeOut $toplevel msecs cmd
```

- You can set an input handler using

```
XtAddInput device cmd
```

- In DtKSH, you can add your own actions, using a generic actions handler

```
ksh_cmd
```

```
# add a callback
XtAddCallback $list singleSelectionCallback
listCB

# set it going
XtRealizeWidget $toplevel
XtMainLoop
```

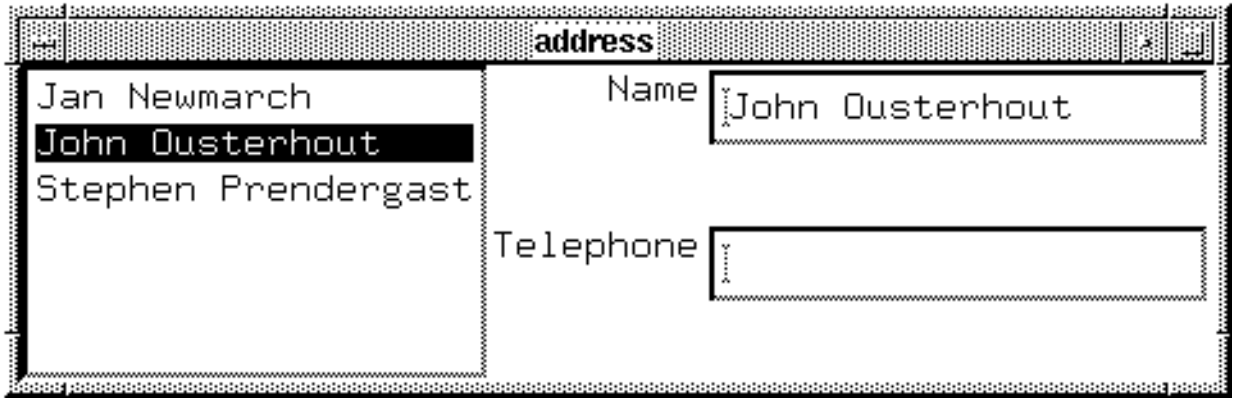
```
XtSetValues $name \
    ‘floattop‘ \
    ‘floatright‘ \
    ‘leftof $Name‘
XtSetValues $telephone \
    topAttachment:attach_position \
    topPosition:50 \
    ‘floatright‘ \
    ‘leftof $Telephone‘

# set values in List
XtSetValues $list \
     items:”Jan Newmarch, John Ousterhout,
Stephen Prendergast” \
    itemCount:3
```

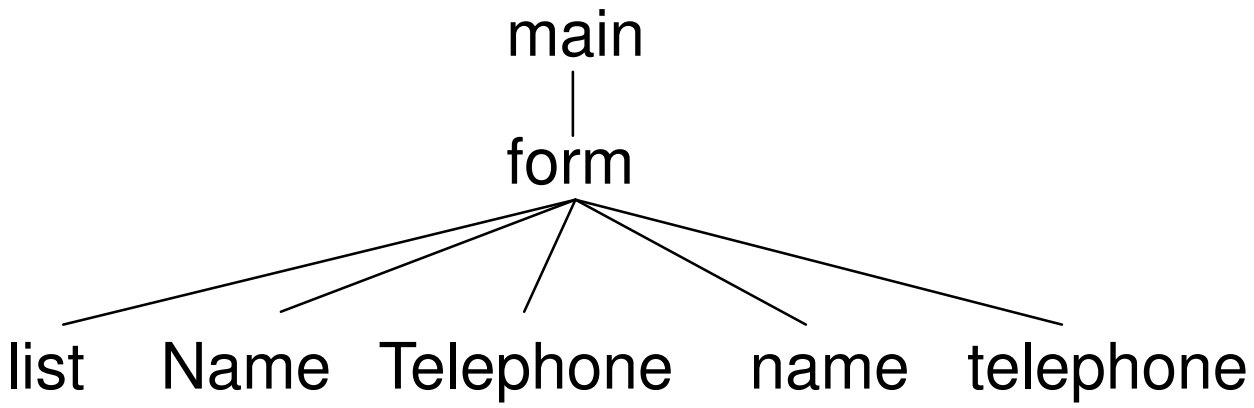
```
XmCreateText -m name $form name
XmCreateText -m telephone $form telephone

# geometry
XtSetValues $list \
    `floatleft` \
    `spanheight`
XtSetValues $Name \
    `floattop` \
    `leftof $list` \
    rightAttachment:attach_opposite_widget\
    rightWidget:$Telephone
XtSetValues $Telephone \
    topAttachment:attach_position \
    topPosition:50 \
    `leftof $list`
```

```
# callback when list item selected
listCB() {
    item=‘listgetsel $list‘
    XtSetValues $name \
        value:”$item”
}
XtAppInitialize toplevel address Address $@

# widgets
XmCreateMainWindow -m main $toplevel main
XmCreateForm -m form $main form
XmCreateList -m list $form list \
    selectionPolicy:single_select
XmCreateLabel -m Name $form Name \
    alignment:alignment_end
XmCreateLabel -m Telephone $form Telephone\
    alignment:alignment_end
```

WKSH Address Book

- A simple address book in the WKSH





- When a callback function is invoked in Motif using C, a structure is passed in as the “call_data” that may contain lots of information:

```
struct {
        int         reason;
        XmString    item;
        int         item_length;
        int         item_position;
} XmListCallbackStruct
```

- Within a DtKSH (not WKSH) callback the same information is available in the variable CB_CALL_DATA:

```
${CB_CALL_DATA.REASON}
$(CB_CALL_DATA.ITEM}
$CB_CALL_DATA.ITEM_LENGTH}
```

WKSH Callbacks

- Any WKSH code can be defined as callback code.

```
XtAddCallback $btn armCallback \
        "echo hi there"
```

- Within a callback, the variable CB_WIDGET contains the widget of the callback function.

Special widgets

- There is a set of convenience functions to manipulate the Text widget.

```
TextOp [-grsGSC] $widget ...
```

with aliases

```
textget, textset, textreplace,
textgetsel, textsetsel, textclearsel
```

- There is a set of convenience functions to manipulate the List widget.

```
ListOp [-adpsgSDitb] $widget ...
```

with aliases

```
listadd, listdel, listsel, ...
```

- There is a full range of functions to draw in the DrawingArea widget.

Geometry management

- WKSH uses the standard Motif geometry management. The Motif widgets are RowColumn, Form, PanedWindow, MainWindow.

```
XmCreateForm form $toplevel "form"
XmCreateLabel label1 $form "label1"
XmCreateLabel label2 $form "label2"
XtSetValues $label1 \
    topAttachment:attach_form
XtSetValues $label2 \
    topAttachment:attach_widget \
    topWidget:$label1
```

- There is a set of convenience functions for Form: under, rightof, floatleft, spanwidth, ...

- It includes the convenience functions ScrolledList, ScrolledText

WKSH Motif functions

- There is a creation function for each of the Motif widgets and gadgets. They are all of the form
  `XmCreate<widget>`
- The set includes the Primitive widgets:
  ArrowButton, CascadeButton, DrawingArea, DrawnButton, Label, List, Scale, Text, ToggleButton.
- It includes the Manager widgets:
  Form, MainWindow, PanedWindow, RowColumn
- It includes the convenience dialogs:
  DialogShell, FileSelectionDialog, FormDialog, InformationDialog, MessageDialog, QuestionDialog, WarningDialog, Working Dialog

WKSH Xt functions

- XtAppInitialize toplevel app–name class–name args
- XtRealizeWidget $widget
- XtMainLoop
- XtSetValues $widget var–args–list
- XtGetValues $widget var–args–list
- Other common Xt functions are included to manage, popup, set sensitivity, check state, etc

WKSH API Structure

- In the WKSH this becomes

```
ActivateCB() { ...
}

XtAppInitialize toplevel ”name” ”class” $@
XmCreateArrowButton –m btn $toplevel ”btn”
XtSetValues $btn arrowDirection:ARROW_UP
XtAddCallback $btn activateCallback \
          ActivateCB
XtRealizeWidget $toplevel
XtMainLoop
```

```
            ActivateCB, NULL)
XtRealizeWidget(toplevel)
XtAppMainLoop(app)
```

C API Structure

- In C this becomes

```
void ActivateCB(Widget w,
     XtPointer call_data,
     XtPointer client_data)
{ ... }

toplevel = XtAppInitialize(&app, Class,
                NULL, 0, &argc, arv,
                NULL, NULL);
btn = XmCreateArrowButton(toplevel, ”btn”,
                NULL, 0)
XtManageChild(btn);
XtVaSetValues(btn, XmNarrowDirection,
                    XmARROW_UP, NULL)
XtAddCallback(btn, XmNactivateCallback,
```

API Structure

- The general structure of the API is e.g.

```
ActivateCB()
{ ... }

XtAppInitialize()
XmCreateArrowButton()
XtManageChild()
XtVaSetValues(..., XmNarrowDirection,
                   XmARROW_UP, ...)
XtAddCallback(..., XmNactivateCallback,
                   ActivateCB, ...)
XtRealizeWidget()
XtAppMainLoop()
```

Xt Overview

- The structure of an Xt–based program is

```
<Callback function definitions>
<Initialise toolkit>
<Create and configure widgets>
     <Create widget>
     <Manage widget>
     <Set resources>
     <Add callbacks>
<Realize widgets>
<Call event loop>
```

Korn shell example

```
# find maximum of a set of numbers
# read from stdin, one number per line

max () {
    if test $1 -gt $2
    then echo $1
    fi
    echo $2
}

read maximum
while read next
do maximum=‘max $maximum $next‘
done
echo ”maximum value is $maximum”
```

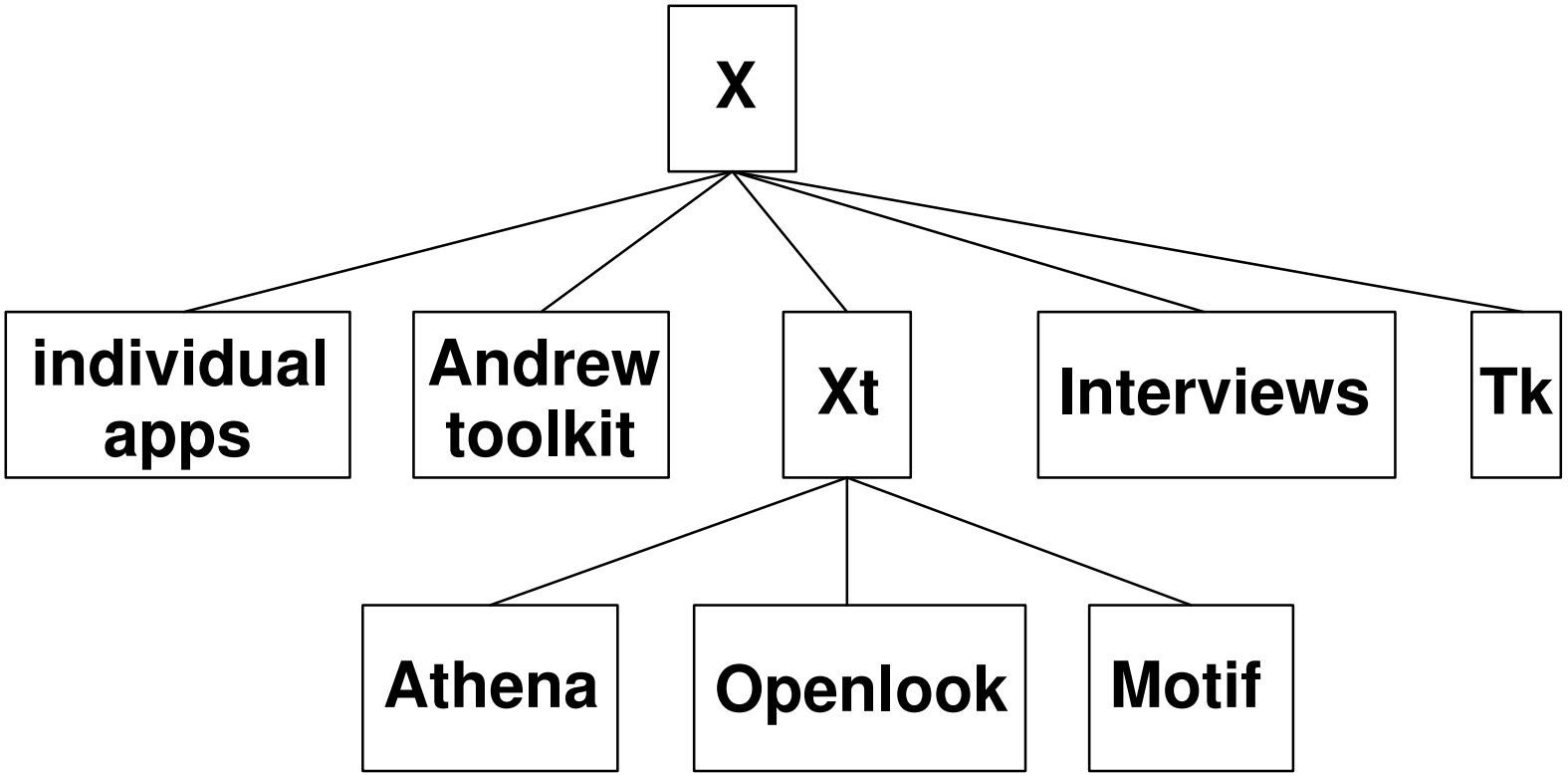
Korn Shell

- Unix shell command language.
- Derivative of Bourne shell.
- Full programming language
  - Assignment statement
  - Conditional statements
  - While, for, until loops
  - Functions
  - Command substitution ‘...‘

Motif

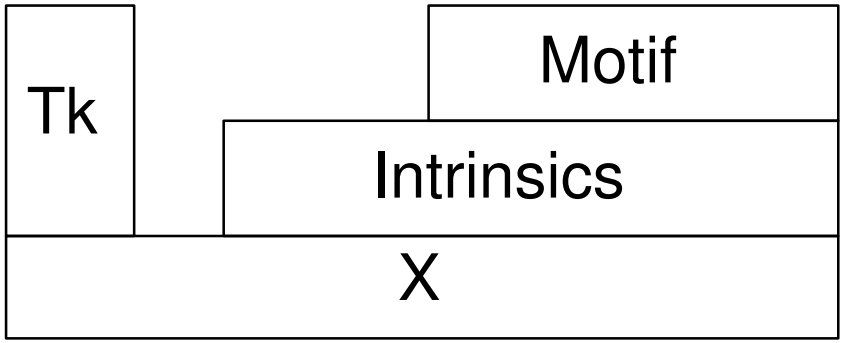
- Motif defines a standard “look and feel”
- “Application Environment Specification” defines the Motif API.
- Essentially a ‘C’ language interface.
- Single implementation by OSF in ‘C’.
- Look and feel of other toolkits attempts to conform to this.
- Alternative bindings have been created:
  - C++
  - Ada
  - Prolog, Poplog, Lisp
  - Korn shell
  - tcl

Toolkits

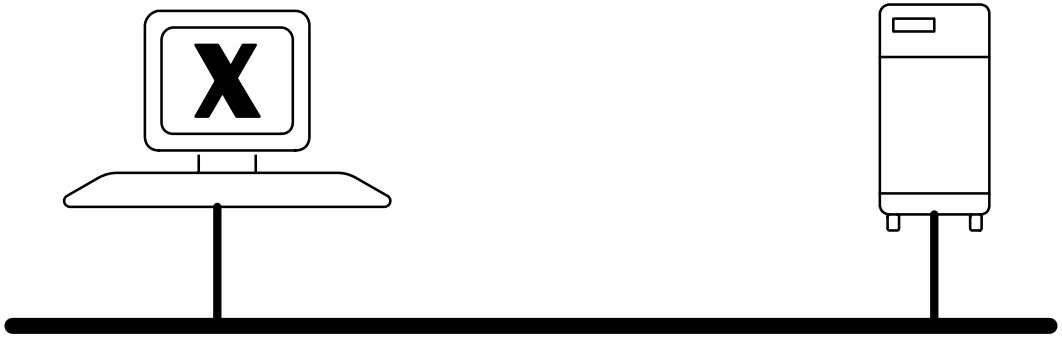

X
individual apps
Andrew toolkit
Xt
Interviews
Tk
Athena
Openlook
Motif

X Window System

- Graphical user toolkit for Unix, VMS.
- Layered model



- Distributed model



Command Languages

A command and response language is a computer language whose primary concern is to invoke system functions, and to convey system responses.

D. Beech, 1979
IFIP Command Languages

- Examples:
  - Shell command languages.
  - Spreadsheet macro languages.
  - Word–processor macro languages.
- Characteristics:
  - Imperative.
  - Interpreted.
  - Weakly typed.

Contents

- Introduction
- WKSH
- Tk
- tclMotif
- Wafe
- New systems
- Implementation

Programming X using Command Languages

Jan Newmarch