Programming with InterViews

Mark Linton
Silicon Graphics
2011 N. Shoreline Blvd.
P.O. Box 7311
Mountain View, CA 94039-7311
linton@sgi.com

Outline

Part 1– Introduction

- Overview
- A simple application (hi mom!)
- Boxes and glue
- Common widgets

Part 2– Graphics

- Rendering
- Fonts
- Colors
- Bitmaps
- Raster images

Outline (cont'd)

Part 3 – Composition

Discretionaries
Compositors
Scrolling
A simple document previewer

Part 4 – Input Handling

Event processing
Update management
Dialogs

Overview

Goals

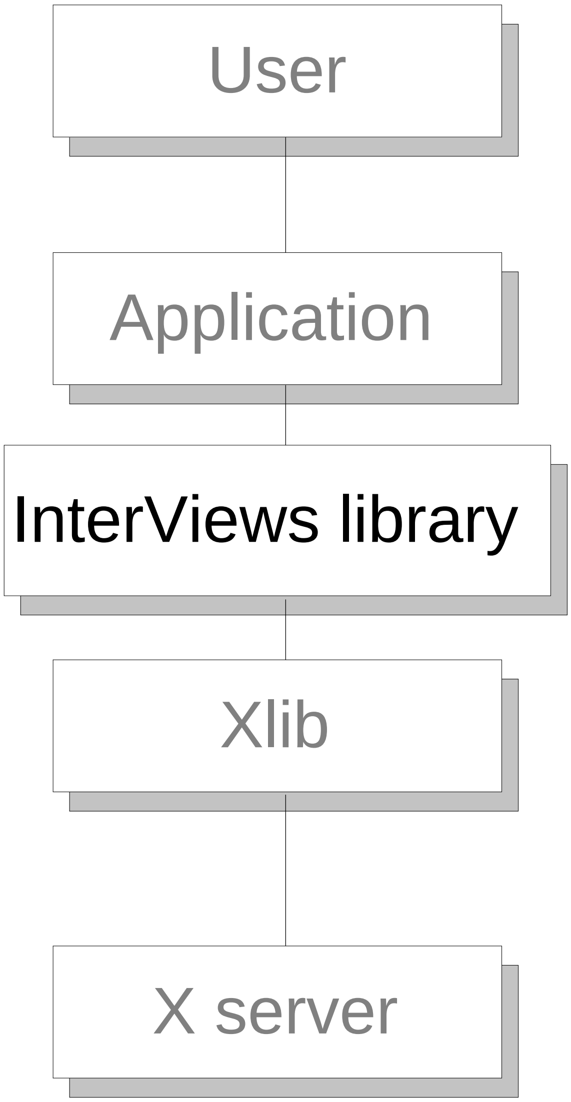
Key features

Differences from other toolkits

System components

Status

High-level toolkit



InterViews Goals

Make user interfaces easier to build

Tools should build applications faster

Need primitives *and* applications

Build real applications

Document, drawing editors

Make reusable components

Composition mechanisms

Leverage technology

C++, Current/future hardware

InterViews = Interactive **Views**

Interactive objects

- Buttons, menus, scrollbars
- Chooser, dialogs, editors
- Multiple views

Graphics objects

- Immediate and structure mode
- Transformations
- Color, font objects

Layout objects

- Sophisticated formatting in toolkit

Graphical editing framework

- Connectivity
- Undo/redo

Natural C++ API

Differences from other toolkits

Layout mechanisms

TeX boxes and glue, discretionaries

Graphics

Transformations (including fonts)
Resolution-independence
Direct color
Structured graphics

Dynamic protocol

Glyphs: lightweight, shareable objects

Native C++

Efficient, object-oriented language

InterViews Organization

Base library

Intrinsic classes (InterViews)
Look+feel classes (IV-look)
X-dependent implementation (IV-X11)
Dispatcher (Dispatch)
Operating system interfaces (OS)

Unidraw library

Applications

doc – document editor
ibuild – interface builder
idraw – drawing editor

Configuration files

Unidraw

Components – objects in drawing
- Connectors – graphical connectivity
- State variables – dataflow

Tools – prototypes/direct manipulation

Commands – undo-able actions

External representations – for domain
- Postscript for all domains

Current status

- Thousands of users world-wide
  - Companies, research labs, universities
- 3.0.1 available via anonymous ftp
- Working on X Consortium standard
- Future work
  - Documents
  - Multimedia
  - Multithreading

Availability

Unrestricted copyright (just like X)

Anonymous ftp to sgi.com
graphics/interviews/3.0.1.tar.Z

Requires >= C++ 2.0, X11R4

Runs on X/Unix platforms
(SGI, HP, Sun, DEC, IBM, ...)

A simple application

Source code for “hi mom!”

Basic types, classes

Building the application

```
#include <IV-look/kit.h>
#include <InterViews/background.h>
#include <InterViews/session.h>
#include <InterViews/style.h>
#include <InterViews/window.h>

int main(int argc, char** argv) {
   Session* session = new Session(
      "Himom", argc, argv
   );
   Style* style = session->style();
   return session->run_window(
      new ApplicationWindow(
         new Background(
            Kit::instance()->label("hi mom!", style),
            style->background()
         )
      )
   );
}
```

Notation

typedef float Coord;
Default units are printer's points
Relative to fonts (typically 75/72)

typedef unsigned int boolean;
static const unsigned false = 0;
static const unsigned true = 1;

enum DimensionName {
Dimension_X, Dimension_Y, Dimension_Z,
Dimension_Undefined
};

#include <InterViews/enter-scope.h>
Define name to **iv***name*

No public or protected data members!

Basic classes

Glyph – an object that draws

Canvas – a 2D place to draw

Window – managed canvas on display

Display – logical input/output devices

Session – main loop coordinator

Kit – object that creates “widgets”

Style – set of <name,value> attributes

Include directories

Intrinsics: <InterViews/*class*.h>

Look+feel: <IV-look/*class*.h>

X-dependent: <IV-X11/*class*.h>

OS-dependent: <OS/*interface*.h>

Dispatching: <Dispatch/*class*.h>

Imakefiles

```
#ifdef  InObjectCodeDir

OBJS = main.o

APP_CCDEFINES =
APP_CCINCLUDES =
APP_CCLDFLAGS =
APP_CCLDLIBS =

Use_libInterViews()
ComplexProgramTarget(himom)

MakeObjectFromSrc(main)

#else

MakeInObjectCodeDir()

#endif
```

Building the application

Default is object files in subdirectory

ivmkmf generates *Makefile*

make Makefiles generates *Makefile* in subdirectory (SGI)

make depend computes dependencies

make builds *a.out*

make install puts *a.out* in installed bin

Layout with boxes

High-level layout specification

Describe format, not position

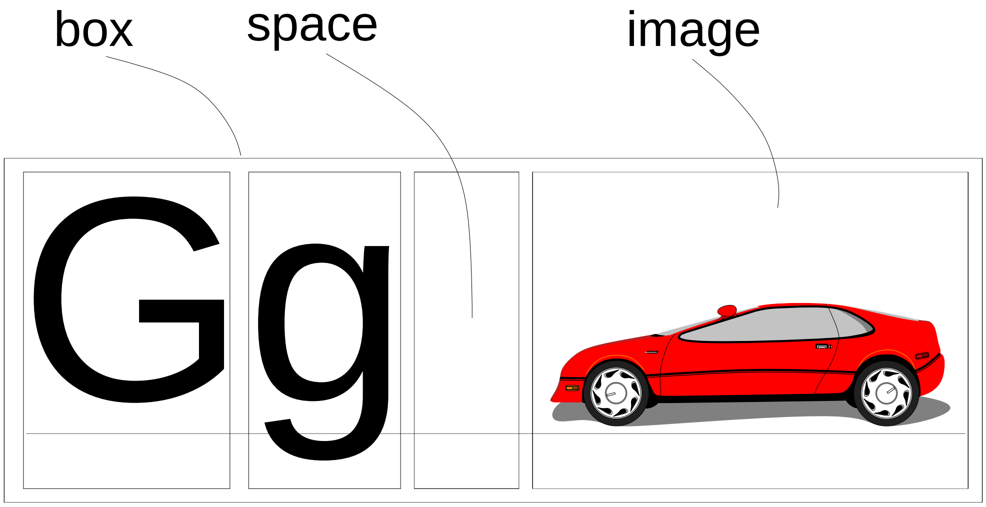
Use document formatting power:
  TeX boxes and glue

Glyphs define “natural size”

Glyphs may “stretch” or “shrink”

Glyphs may define an “alignment”
  Location of origin relative to size

Think of glyphs as characters



Requisition – what a glyph wants

Requirement per dimension

Natural, stretch, shrink, alignment

Returned by Glyph::request

Allocation – what a glyph gets

Allotment per dimension

Origin, span, alignment

Passed to Glyph::draw

Boxes

Use a layout to arrange components

Tile layout – proportional stretch/shrink

Align layout– line up origins

Superpose layout – multiple layouts (usually one per dimension)

LRBox – Tile in X, Align in Y

TBBox – TileReversed in Y, Align in X

Box Example 1

```
// start with hi mom code
const Font* f = style->font();
const Color* fg = style->foreground();
return session->run_window(
  new ApplicationWindow(
    new Background(
      new LRBox(
        new Character('g', f, fg),
        new Character('o', f, fg),
        new Character('o', f, fg),
        new Character('d', f, fg),
        new Character('b', f, fg),
        new Character('y', f, fg),
        new Character('e', f, fg)
      ),
      style->background()
    )
  )
);
```

Box Example 2: Add a stencil (twice)

```
#include <InterViews/Bitmaps/hand.bm>
  . . .
  new LRBox(
    new Stencil(
      new Bitmap(
        hand_bits, hand_width, hand_height
      ),
      fg
    ),
    new Stencil(
      new Bitmap(
        hand_bits, hand_width, hand_height,
        hand_x_hot, hand_y_hot
      ),
      fg
    ),
    . . .
```

Box Example 3: Nested LRBoxes

```
new LRBox(
  new LRBox(
    new Character('g', f, fg),
    new Character('o', f, fg),
    new Character('o', f, fg),
    new Character('d', f, fg)
  ),
  new LRBox(
    new Character('b', f, fg),
    new Character('y', f, fg),
    new Character('e', f, fg)
  )
)
```

Glue for stretching and shrinking

```
new LRBox(
   new Label("good", f, fg),
   new HGlue(0, fil, 0),
   new Label("bye", f, fg)
)
```

TileReversed can be surprising

```
// This box isn't stretchable!
new TBBox(
  new Label("good", f, fg),
  new VGlue(0, fil, 0),
  new Label("bye", f, fg)
)

// This box is stretchable in the middle,
// but not at top.
new TBBox(
  new VGlue(0, fil, 0),
  new Label("good", f, fg),
  new VGlue(0, fil, 0),
  new Label("bye", f, fg)
)
```

Tuning alignment

TileReversed computes request from minimum, maximum sizes, alignments

Default alignment is 0 (beginning)

Zero alignment fine for Tile, but often wrong from TileReversed

No room for glue to stretch

Solution is to adjust alignments

```
new TBBox(
   new VCenter(new Label("good", f, fg), 1.0),
   new VGlue(0, fil, 0),
   new Label("bye", f, fg)
)
```

VCenter changes the vertical alignment

Top of label aligns with top of box

Common Widgets

Kits
Coordinating a common look+feel

Buttons
Telltales and Actions,
Push, toggle, radio

Menus
Menubars, toggle items, radio items

Kit

Object that creates UI objects

Buttons, beveled frames, menus, scrollbars, labels, common cursors, ...

Hides subclass/instance choices

Example: Button* Kit::push_button

One per application

Returned by Kit::instance

Subclasses for specific look+feel

Motif, OpenLook, SGI-Motif, Monochrome

Buttons

Telltale – glyph with multiple looks

{ is_enabled, is_visible, is_active, is_chosen }

Action – object to execute
ActionCallback(T) – C++ callback

Button(Telltale*, Action*)

Input action sets telltale state

Release executes action

Defining an action callback

```
class App {
public:
  void msg();
};

declare(ActionCallback,App)
implement(ActionCallback,App)

void App::msg() {
  printf("hi mom!\n");
}

App* a = new App;
  ...
... new ActionCallback(App)(a, &App::msg) ...
```

Creating a push button

```
int main(int argc, char** argv) {
   Session* session = new Session(
      "Himom", argc, argv
   );
   Style* style = session->style();
   Kit* kit = Kit::instance();
   App* a = new App;
   return session->run_window(
      new ApplicationWindow(
         new Background(
            new Margin(
               kit->simple_push_button(
                  "Push me", style,
                  new ActionCallback(App)(a, &App::msg)
               ), 10.0
            ), style->flat()
         )
      )
   );
}
```

Using a different button look

```
Raster* rast = TIFFRaster::load(argv[1]);
if (rast == nil) {
  ... error message ...
}

return session->run_window(
  new ApplicationWindow(
    new Background(
      new Margin(
        kit->push_button(
          new Image(rast), style,
          new ActionCallback(App)(a, &App::msg)
        ), 10.0
      ), style->flat()
    )
  )
);
```

Toggle and radio buttons

```
kit->simple_toggle_button(
  "Toggle me", style,
  new ActionCallback(App)(a, &App::msg)
)

TelltaleGroup* group = new TelltaleGroup;

new TBBox(
  new VCenter(
    kit->simple_radio_button(
      group, "One", style, nil
    ),
    1.0
  ),
  kit->simple_radio_button(
    group, "Two", style, nil
  )
)
```

Menus

List of items: telltale, {action, menu}

Menu alignment defines submenu position

```
void Menu::add_item(Telltale*, Action*)
void Menu::add_item(Telltale*, Menu*)
```

```
Menu* Kit::menubar(Style*)
Menu* Kit::pulldown(Style*)
```

```
Telltale* Kit::menubar_item(Glyph*, Style*)
Telltale* Kit::menu_item(Glyph*, Style*)
Telltale* Kit::toggle_menu_item(Glyph*, Style*)
Telltale* Kit::radio_menu_item(
  TelltaleGroup*, Glyph*, Style*
)
```

Menu example

```
struct CmdInfo;

class App {
public:
   void open(), save(), quit();
   void cut(), copy(), paste();
   void black(), red(), green(), blue();

   Menu* menubar(CmdInfo*, Kit*, Style*);
private:
   Menu* pulldown(CmdInfo*, int, Kit*, Style*);
};

declare(ActionCallback,App)
implement(ActionCallback,App)
```

Menu example (cont’d)

```
struct CmdInfo {
   const char* str;
   ActionMemberFunction(App)* func;
   CmdInfo* submenu;
   int options;
};

CmdInfo filemenu[] = {
   { "Open", &App::open },
   { "Save", &App::save },
   { "", nil },
   { "Quit", &App::quit },
   { nil }
};

CmdInfo bar[] = {
   { "File", nil, filemenu, 0 },
   { "Edit", nil, editmenu, 1 },
   { "Color", nil, colormenu, 2 },
   { nil }
};
```

Menu example (cont’d)

```
void App::open() { printf("open\n"); }
void App::save() { printf("save\n"); }
void App::quit() { Session::instance()->quit(); }

void App::cut() { printf("cut\n"); }
void App::copy() { printf("copy\n"); }
void App::paste() { printf("paste\n"); }

void App::black() { printf("black\n"); }
void App::red() { printf("red\n"); }
void App::green() { printf("green\n"); }
void App::blue() { printf("blue\n"); }
```

Menu example (cont'd)

```
Menu* App::menubar(CmdInfo* info, Kit* k, Style* s) {
  Menu* m = k->menubar(s);
  for (CmdInfo* i = info; i->str != nil; i++) {
    m->add_item(
      k->menubar_item(k->fancy_label(i->str, s), s),
      pulldown(i->submenu, i->options, k, s)
    );
  }
  // disable save
  m->menu(0)->telltale(1)->set(
    Telltale::is_enabled, false
  );
  return m;
}
```

Menu example (cont’d)

```
Menu* App::pulldown(
  CmdInfo* info, int opt, Kit* k, Style* s
) {
  Menu* m = k->pulldown(s);
  TelltaleGroup* group = nil;
  for (CmdInfo* i = info; i->str != nil; i++) {
    if (i->str[0] == ’\0’) {
      m->add_item(k->menu_item_separator(s));
    } else {
      // Build the telltale
      // Add the item
    }
  }
  return m;
}
```

Menu example (cont'd)

```
// Build the telltale
Telltale* t;
Glyph* g = new RMargin(
  k->fancy_label(i->str, s), 0.0, fil, 0.0
);
switch (opt) {
default:
  t = k->menu_item(g, s);
  break;
case 1:
  t = k->toggle_menu_item(g, s);
  break;
case 2:
  if (group == nil) {
    group = new TelltaleGroup;
  }
  t = k->radio_menu_item(group, g, s);
  break;
}
```

Menu example (cont'd)

```
// Add the item
if (i->func == nil && i->submenu != nil) {
   m->add_item(
      t, pulldown(i->submenu, i->options, k, s)
   );
} else {
   m->add_item(
      t, new ActionCallback(App)(this, i->func)
   );
}
```

Summary for Part 1

System overview

Glyph – base for interface objects
(glyph ~ character)

Kit – create concrete GUI objects

Using boxes and glue for layout

Creating buttons and menus