<table>
<thead>
<tr>
<th>Time</th>
<th>Session</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>9:30</td>
<td><strong>Fundamentals 1:</strong></td>
<td>What is X? platform independence, network and client/server operation, X protocol, event model, windows</td>
</tr>
<tr>
<td>10:30</td>
<td>Practical 1:</td>
<td>first Motif program</td>
</tr>
<tr>
<td>11:00</td>
<td><strong>short break</strong></td>
<td></td>
</tr>
<tr>
<td>11:10</td>
<td><strong>Fundamentals 2:</strong></td>
<td>X Motif, X intrinsics and X lib, role of libraries, naming conventions</td>
</tr>
<tr>
<td>11:40</td>
<td><strong>Fundamentals 3:</strong></td>
<td>Structure of Motif programs, creating and realising widgets, linking call-backs</td>
</tr>
<tr>
<td>12:10</td>
<td>Practical 2:</td>
<td>pop-up windows, using call-backs</td>
</tr>
<tr>
<td>12:30</td>
<td><strong>lunch</strong></td>
<td></td>
</tr>
<tr>
<td>1:30</td>
<td>Practical 3:</td>
<td>looking at the elements: labels, buttons, forms, shells, modifying examples</td>
</tr>
<tr>
<td>3:20</td>
<td><strong>short break</strong></td>
<td></td>
</tr>
<tr>
<td>3:30</td>
<td><strong>Fundamentals 4:</strong></td>
<td>the widget class hierarchy, resource values parent/child relationship and run-time hierarchy widgets and gadgets, geometry management</td>
</tr>
<tr>
<td>4:10</td>
<td>Practical 4:</td>
<td>constructing a main window and menu bar</td>
</tr>
<tr>
<td>4:30</td>
<td><strong>session ends</strong></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>Activity</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>---------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>9:30</td>
<td>Practical 5:</td>
<td>Pull down menus</td>
</tr>
<tr>
<td>10:10</td>
<td>Practical 6:</td>
<td>A full application</td>
</tr>
<tr>
<td>11:00</td>
<td><strong>short break</strong></td>
<td></td>
</tr>
<tr>
<td>11:10</td>
<td>Reading:</td>
<td>A look at Motif books</td>
</tr>
<tr>
<td>11:30</td>
<td>Practical 7:</td>
<td>Adding real application functionality</td>
</tr>
<tr>
<td>12:30</td>
<td><strong>lunch</strong></td>
<td></td>
</tr>
<tr>
<td>1:30</td>
<td>Fundamentals 5:</td>
<td>Motif style guide and usability</td>
</tr>
<tr>
<td>2:00</td>
<td>Practical 8:</td>
<td>Looking at other widgets</td>
</tr>
<tr>
<td>3:00</td>
<td><strong>short break</strong></td>
<td></td>
</tr>
<tr>
<td>3:10</td>
<td>Practical 9:</td>
<td>More complex applications</td>
</tr>
<tr>
<td>4:15</td>
<td><strong>review session</strong></td>
<td></td>
</tr>
<tr>
<td>4:30</td>
<td><strong>course ends</strong></td>
<td></td>
</tr>
</tbody>
</table>
Motif Reading


N.B. Rebadged copies of 5, 6 and 7 may be part of a commercial Motif distribution.
Motif

Part I
Fundamentals
Motif Fundamentals 1

- what is X?
- a little history
- network and client/server operation
- the X protocol
- windows everywhere
- X event model
What is X?

X is a –

• non-proprietary
• platform independent
• client/server
• network transparent
• bitmap window system

X is not –

• a UNIX window manager
• a window manager
• only for UNIX
A Little History

Development of X

1983     Digital Western Research Laboratory
          W a prototype window system ported to VS100s terminals

1984     MIT – Project Athena (sponsored by DEC & IBM)
          need a UNIX bitmap window system
          building on W, but modified significantly (sync → async)
          W becomes X!

1986/7    X in version 10
           ports to HP, Apollo, Sun, IBM
           DEC WRL develop version 11 for public distribution

1988     MIT X Consortium founded
           aim – to develop X and maintain its independence

Motif

• developed by Open Software Foundation (OSF)
  (a consortium including HP, Digital, IBM developing interoperability standards)

• standard for:
  user interface appearance and behaviour
  programmers API

• modelled on IBM Common User Access (CUA)
client/server

- X server  –  on workstation, manages screen
- X client  –  the application

? the wrong way round?

- database  –  shared resource  –  data
  –  server manages  –  data

- X  –  shared resource  –  display
  –  server manages  –  display
Client / Server

X server

- on UNIX workstation
- on dedicated X terminal
- on PCs
- on Macs

  - manages one or more screens
  - may ‘control’ whole screen or operate with native window manager

X client

- anywhere! even transcontinental

Special case

- client and server on same machine
- always run as separate processes
X Protocol

X standard
    • what goes down the wire

Any wire
    • Internet TCP/IP
    • DECNet
    • even over a modem!
★ any duplex byte-stream connection

client → server: request  e.g., put “Hello World” on screen
server → client: events   e.g., user typed “X”

✗ network is bottleneck
    • specification is core of X
    • asynchronous
    • beware of delays

✔ protocol is extensible
    • e.g., Motif extensions for workspaces
X ‘windows’

\[
\begin{align*}
\text{X window} & = \text{rectangle on screen} \\
\text{screen} & = \text{actual glass thing} \\
\text{display} & = \text{workstation} \\
& \quad \text{possibly many screens} \\
& \quad \text{but only one mouse & keyboard}
\end{align*}
\]

lots of windows:
- each button, scrollbar, menu, icon etc.

raw X is very basic
- output: draw lines or text
- input: react to user actions
- resources: manage colour-maps, fonts
- structure: hierarchy of windows
  - children clipped by parents
X is not a window manager

window manager

- separate but special client
- manages window controls:
  - title bar, zoom box, resizing, generic menu
- can run anywhere
  - same place as server
  - same place as client
  - somewhere else entirely

X standard defines protocol: client ↔ WM

Possible to bypass window manager . . .
  . . . but not a good idea!

Different styles of window manager
  including Motif window manager
  ( N.B. Motif applications do not need MWM )
X Events

Source

• user activities:
  mouse clicks, keyboard, mouse movement
• window manager events:
  keyboard focus, resize
• inter-application communication:
  cut/paste, drag’n’drop

Control

client: registers interest in specific events
server: selects relevant events
        merges into a single event stream

Content

type: keyboard, mouse press etc.
window: where event occurred
sync & timestamp: N.B. asynchronous
mouse position: ditto!
other info: e.g., which key

Naming

    heavy use of IDs – window, fonts, colour-maps ...
**X Event Loop**

**Client code:**

```
loop forever
  read next event
  switch ( event.type ) {
    case mouse press:
      if ( event.window == ... )
        . . .
    case key press:
      case . . .
  }
end loop
```

**The Good News**

Motif manages this for you!

**The Bad News**

Need to learn event based programming
Different levels

- X protocol
- Xlib – raw X  
  X standard API
- Xt – intrinsics  
  objects oriented features  
  + event management
- Xm – Motif toolkit  
  buttons, menus etc.

Naming

  XDrawLine  
  XtVaCreateManagedWidget  
  xmLabelGadgetClass
Xlib

programmers API for protocol
  • should never use protocol directly
  • except for building servers etc.

raw X
  • open displays
  • create windows
  • draw graphics: lines, pixels and text

very basic
  ⇒ need toolkits to add basic widgets
e.g., Motif

Xlib calls rarely needed when using Motif

exceptions
  • drawing lines, graphics etc.
  • understanding event structure
Motif compliant toolkit

N.B. Motif is the standard not the toolkit

Defines common GUI widgets

- scroll bars
- buttons
- text areas
- menus
- dialogue boxes

Does a lot of work for you!

- handles a lot of user interaction
- manages layout of windows
- automates resizing, redrawing etc.

... but you can control it if you want to!
Xt – X intrinsics

defines an OO-like layer for widgets

handles common things:

- creating widgets
  
  \begin{verbatim}
  XtVaCreateManagedWidget( "Open",
    xmPushButtonClass, button_area, NULL )
  \end{verbatim}
  (defined in Xm)

- resources for widgets
  
  attributes such as position, size, etc.
  e.g., XmNwidth, XmNlabelString

- setting/getting resources
  
  \begin{verbatim}
  XtSetValue( widget, XmNresName, resValue )
  \end{verbatim}

- reading resource databases

- the event loop
  
  \begin{verbatim}
  XtAppMainLoop( ... )
  \end{verbatim}

N.B. Motif is largely Xt class definitions
Structure of Motif Program

Main routine

1. initialise toolkit
2. create widgets
3. ( add callbacks where necessary )
4. realise widgets
5. enter main event loop

Event routines

• user defined functions
• called when specific events happen
• different ones for different types of events
Initialisation

XtVaAppInitialise( ... )

1. opens connection to X server
2. parses command line arguments
   (handles standard X arguments such as window position)
3. set-up default resources (e.g., fontname)
4. opens resource databases
5. creates a top level ‘shell’ widget

☆ can do it bit by bit for special purposes
Creating Widgets

XtVaCreateManagedWidget( name, class, parent,  
  resName1, resVal1, ..., NULL )

name – your name for the widget  
  used to link to resources  
  e.g., “form”

class – widget class defined by Motif  
  e.g., XmFormWidgetClass

parent – existing widget often contains child

remaining arguments define resource values  
  e.g., XmNwidth, 500,

N.B.
  • creates the internal data structure  
  • does not put anything on screen  
  • does not create X ‘window’

Alternative form of call uses a list of resource arguments
Realising Widgets

\[ \text{XtRealiseWidget}( \text{topWidget} ) \]

At this point

1. an X window is created for the widget
2. all its children are realised
3. where appropriate windows are displayed

Terminology:

- realised – an X window exists
- managed – parent handles positioning (geometry)
- mapped – whether or not widget is displayed

N.B. children can be added after a widget is realised

But:

- may be computationally expensive
- appearance can be strange

OK on occasions (see examples)
Adding Callbacks

Many events managed within Motif
  e.g., text entry, menu selection

but need to respond to application specific events
  e.g., action on button press

⇒ need to call application code when event occurs

\texttt{XtAddCallback( \textit{widget, callback-type, func, my-data} )}

- \texttt{widget} – an existing widget such as a button
- \texttt{type} – a callback resource name
  which type of event to respond to
  e.g., XmNactivateCallback
- \texttt{func} – pointer to C function defined by you
  e.g., quit_func
- \texttt{my-data} – an integer or pointer to your data
  passed on to your callback
Defining Callbacks

When the relevant event happens your function is called

The callback function definition:

```c
void quit_func( widget, my-data, event-data )
```

- **widget** – where the event occurred
- **my-data** – the integer or pointer passed in the call to `XtAddCallback`
- **event-data** – the X event structure which caused the callback

Callback functions may:

- update application data
- exit the program (cleanly!)
- add/delete/modify the Motif widgets
widget sub-class hierarchy

different kinds of widget

widget run-time hierarchy

gadgets
Widget Sub-Class Hierarchy

Sub-class hierarchy based on common behaviour

. . . sort of

Two major types of widgets

primitive — small components
  e.g., button, text area etc.

composite — areas in which other widgets fit
  e.g., scrolled windows, form

N.B. primitive widgets can have children too

In addition, Motif has convenience functions
these generate:

- groups of widgets
- specially configured widgets

  e.g., XmCreateMenuBar

Some widgets create children of their own

  e.g., xmMainWindowClass
Primitive widgets

Label – read-only text label
various buttons as sub-classes
N.B. not true kind-of relationship

CascadeButton, DrawnButton, PushButton, ToggleButton

TextLabel – multi-line text area

TextField – single-line text area

ArrowButton – button with arrow on it!

Separator – used in menus

Scrollbar – used in scrollable windows

List – usually created by XmCreateScrolledList

N.B. string for label is compound string
    can contain multiple lines, fonts, sizes etc.

Also lots of convenience widgets:
    combo box, menu bar, pull down menu ...
Managed widgets arrange children in different ways

- **RowColumn** – left/right, top/bottom or grid
- **DrawingArea** – for graphics
- **Frame** – boxes its child
- **BulletinBoard** – application defined layout
  also several useful sub-classes, inc.
- **Form** – complex arrangements
- **Scale** – ‘primitive’ like widget
  set/show values
  children are ticks
- **ScrolledWindow** – adds scrollbar
  with sub-class (code sharing, not kind of)
- **MainWindow** – adds menubar etc.
  template for many applications
Composite widgets – shells

Shells interact with the window manager

Their single child is the real application

TopLevelShell & ApplicationShell
  - for application windows

TransientShell & DialogueShell
  - for pop-ups

MenuShell  - no window manager controls

For simple applications create top level shell with:
  top_level = XtVaAppInitialise( ... )

or Xm convenience routines
Run-Time Hierarchy

Parent/child relationship between widgets

Normally closely matches parent/child for X windows

exceptions

– dialogues, pop-up and tear-off menus

N.B. very different from sub-class hierarchy!
Gadgets

Gadgets rather like primitive widgets:

- ArrowButtonGadget
- LabelGadget
- CascadeButtonGadget
- PushButtonGadget
- ToggleButtonGadget
- SeparatorGadget

But have no X window attached

Gadgets don’t handle their own events

handled by manager widget that contains them

not full set of resources

e.g., no resource for colour

Why use them?

lots of widgets ⇒ lots of X windows

No X window ⇒ more efficient

But X implementations have improved

windows are cheap
Motif is a style not a toolkit

Xm – simply easy way to be Motif

Keeping to style ⇒ consistency

Xm does a lot . . .
  . . . but you have to work too
Reading the Style Guide

Aimed at different people:

- application designers
- widget builders
- toolkit designers
- window manager designers

Most guidelines handled by toolkit/window manager

Relevant parts for application designer:

1. user interface design principles
2. application design principles
3. internationalisation

+ remember timing problems!
User Interface Design Principles

General principles and good advice for GUI designers

• adopt the user’s perspective

• give the user control
  flexibility, progressive disclosure

• use real-world metaphors
  direct manipulation
  rapid (consistent) response !!
  output → input

• keep interfaces natural
  easy navigation, natural colours

• keep interfaces consistent

• communicate application actions to the user
  feedback, errors, destructive acts

• avoid common design pitfalls
  the process and product
Application Design Principles

Some aspects covered or made easy by Motif toolkit

. . . but you do have to do some work

Which components to use

- menu vs. dialogue
- check box vs. list

etc. ...

Standard menu items: File and Help

Use the right dialogue box!

Principles for arranging screens and menu items

Interaction principles:

- indications of action
- feedback
- user flexibility (customisation)
Part II
Practical
Practical 1

Files:

Makefile
XDefs.h
hello.c

Single window with “hello world” and quit button
Practical 2/3

File:

hello.c

Single window with “hello world” and quit button, and pop-up window launched by “Press me” button.
Practical 4

Files:

hello.c

As practical 2/3, but with quit on menubar rather than on button. Uses MainWindow widget to manage menubar.
This is a small database application. It is written to exemplify different aspects of Motif, not as an example of good interface design.

The application is split into two halves. The front end (db.c) handles the user interface and the back end (db_lib.c) reads and writes the database and performs database searches. The header file db_lib.h defines the programmer's interface between the front end and back end.

This general structure is recommended for any user-interface design project. Obviously in a larger project there would be many files for both back end and front end, but the separation can be maintained. This allows the user interface to be modified easily and also allows different implementations of the back-end functionality. For example, the file db_lib.c could be modified to use a real database engine, rather than a C array. Because of the strong separation this would entail no changes to the user interface.
blank for your notes
Motif

blank for your notes
blank for your notes