CISC 839
Software Engineering of Usable Computing Systems

T.C. Nicholas Graham
http://stl.cs.queensu.ca/~graham/cisc836

Software Architectures for Interactive Systems

- Architectures at different levels
  - Conceptual Architecture
    - High-level implementation plan
    - Closer to design
  - Implementation Architecture
    - How the code is actually constructed

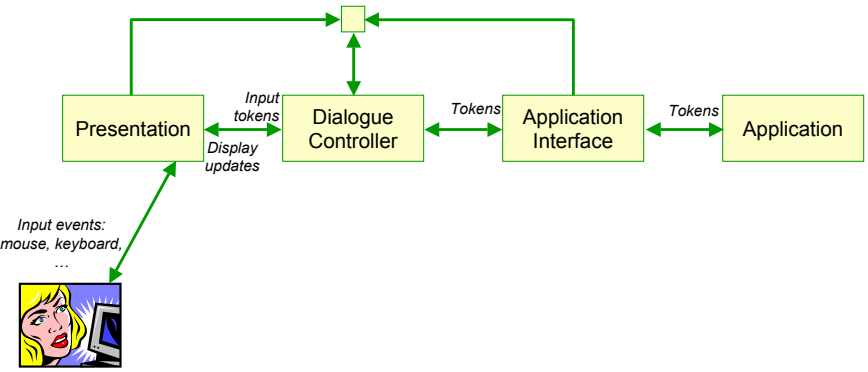
Software Architectures for Interactive Systems

- We will consider
  - Architectures for standard PC applications
  - Architectures for groupware
  - Architectures for mobile applications

Reference Architectures

- Ideal architecture
  - Shows all elements of all architectural approaches
  - Typically not actually used for implementation
  - Helpful for description of architectures, actually systems

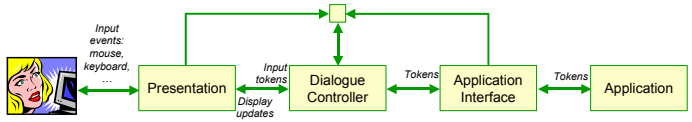
Seeheim Model



Source: Pfaff et al., 1985

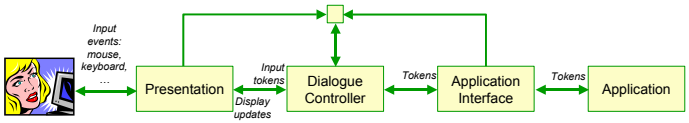
Seeheim Model

- Presentation
  - Responsible for handing raw user inputs
    - Mouse motion, mouse button clicks, key clicks, etc.
    - Provides these events to dialogue controller once processed
      - Mouse events, keyboard events, etc.



Seeheim Model

- Presentation
  - Responsible for presenting display updates to user
    - Interprets high-level commands such as "drawline", "drawstring"
    - Renders updates on display device



Programming Presentation

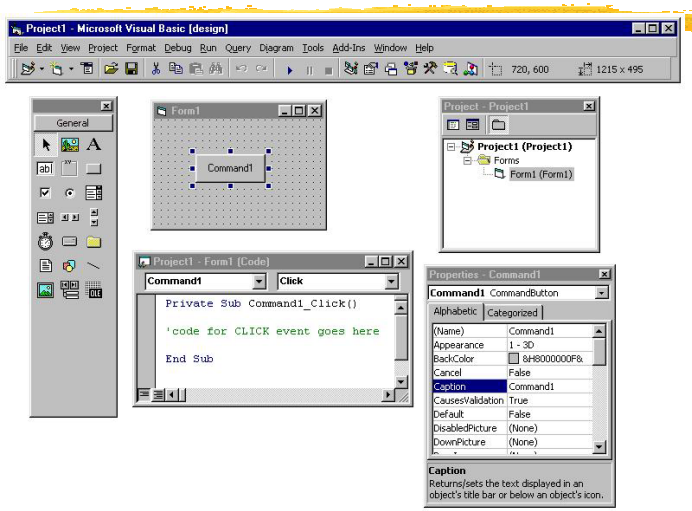
- Library approach
  - Xlib (Unix)
  - GDI (Windows)
- Interface builder
  - Drag and drop interface components

Example: GDI Programming in C++

```
void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);

    //
    // Initialize the device context.
    //
    dc.SetMapMode (MM_LOENGLISH);
    dc.SetTextAlign (TA_CENTER | TA_BOTTOM);
    dc.SetBkMode (TRANSPARENT);

    //
    // Draw the body of the ruler.
    //
    CBrush brush (RGB (255, 255, 0));
    CBrush* pOldBrush = dc.SelectObject (&brush)
    dc.Rectangle (100, -100, 1300, -200);
    dc.SelectObject (pOldBrush);

    //
    // Draw the tick marks and labels.
    //
    for (int i=125; i<1300; i+=25) {
        dc.MoveTo (i, -192);
        dc.LineTo (i, -200);
    }

    for (i=150; i<1300; i+=50) {
        dc.MoveTo (i, -184);
        dc.LineTo (i, -200);
    }

    for (i=200; i<1300; i+=100) {
        dc.MoveTo (i, -175);
        dc.LineTo (i, -200);

        CString string;
        string.Format (_T ("%d"), (i / 100) - 1);
        dc.TextOut (i, -175, string);
    }
}
```

Source: Jeff Prosise, Programming Windows with MFC, Microsoft Press, 1999

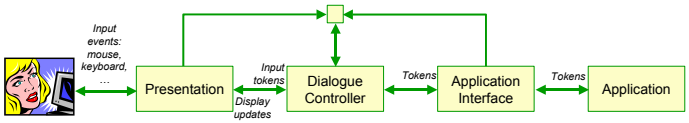
Example: Presentation Design in Visual Basic



Source: Gary Beene's Visual Basic Information Center, http://www.vbinformation.com/tut-ide.htm

Seeheim Model

- Dialogue Controller
  - Processes input tokens to produce higher level tokens
  - Example: sequence of mouse movements, clicks invoke a menu operation
    - Input tokens: mouse motion, mouse click
    - Output token: "File|Save"



Dialogue

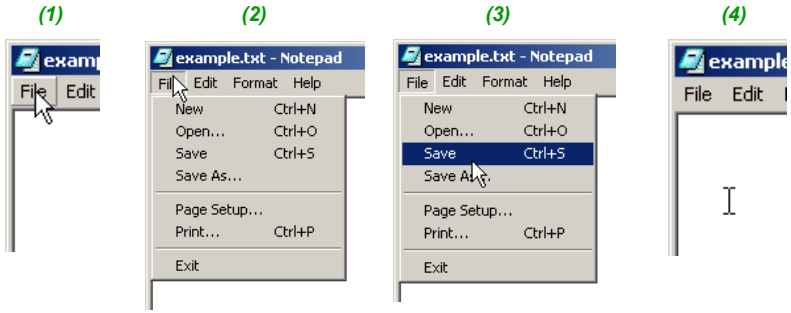
- Interchange between human and computer to perform some task
- Examples
  - Unix file transfer (ftp)
  - Selecting an entry from a menu

Example Dialogue

```
tivoli<175> ftp ftp.cs.yorku.ca
Connected to wolf.cs.yorku.ca.
220-ftp.cs.yorku.ca FTP server (CS.YorkU.Ca 3.93) ready.
220-Report any problems to Sysadm@cs.yorku.ca.
220-Note: Anonymous FTP incoming requires setting "account"
220 Note: to appropriate email address for local user.
Name (ftp.cs.yorku.ca:graham): anonymous
331 Guest login ok, send email address ( eg "your.name@tivoli") as password.
Password: *****
230 Guest login ok, access restrictions apply.
ftp>
```

Black text is initiated by computer. Red text is initiated by user.

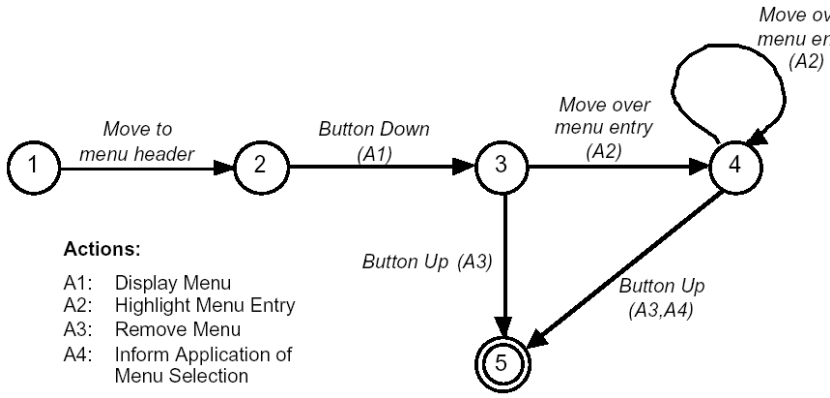
Example Dialogue



Programming Dialogues

- Usually by hand
  - Event handlers
- Transition diagrams (Finite state machines)
- Grammars

Transition Diagram for Menu Selection Dialogue



Grammar for Menu Selection Dialogue

selectFromMenu ::= moveToMenuHeader
ButtonDown **A1**
finishMenu

finishMenu ::= ButtonUp **A3**
| { MoveOverMenuEntry **A2** }$^{+}$
ButtonUp **A3 A4**

Actions:

A1: Display Menu
A2: Highlight Menu Entry
A3: Remove Menu
A4: Inform Application of Menu Selection

Grammar and FSM Approaches

- Poor handling of error conditions
- Poor scalability
- Poor handling of concurrent/interleaved dialogues

Event Handlers in Microsoft Foundation Classes (MFC)

```
#include <afxwin.h>
#include "Ruler.h"

CMyApp myApp;

BOOL CMyApp::InitInstance ()
{
    m_pMainWnd = new CMainWindow;
    ...
}

BEGIN_MESSAGE_MAP (CMainWindow, CFrameWnd)
    ON_WM_PAINT ()
END_MESSAGE_MAP ()

CMainWindow::CMainWindow ()
{
    Create (NULL, _T ("Ruler"));
}

void CMainWindow::OnPaint ()
{
    CPaintDC dc (this);

    ...
    }
```

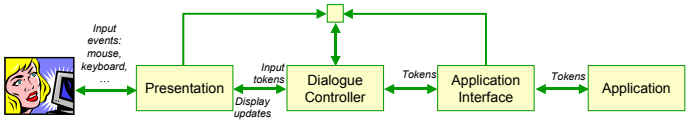
Source: Jeff Prosise, Programming Windows with MFC, Microsoft Press, 1999

Event-based Approaches

- Flow of control of dialogue obscured

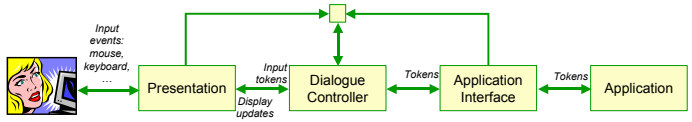
Seeheim Model

- Application Interface
  - Forms gateway to implementation
  - May involve
    - Routing events to correct part of application
    - Interpreting tokens as application calls
      - E.g., application is a database – requires converting application events into ODBC events

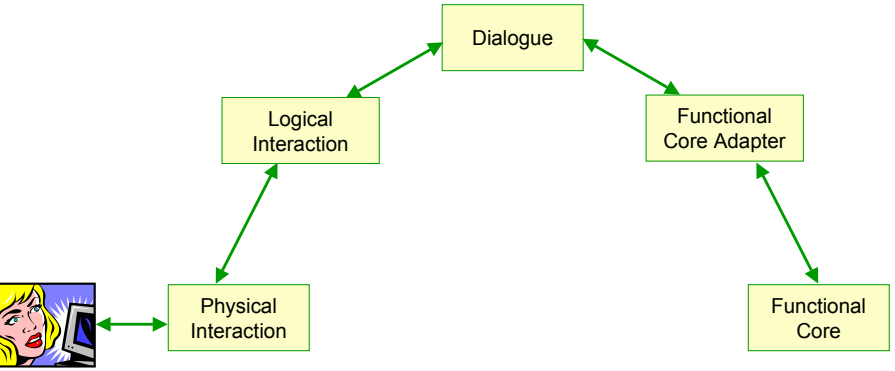


Seeheim Model

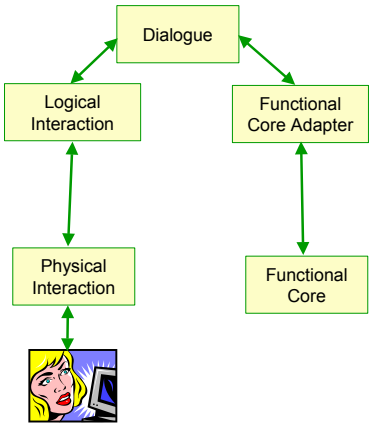
- Application
  - May be tightly bound to user interface
  - May be written in different language
    - E.g., Visual Basic:
      - Typical deployment: UI written in VB, application written in C++



ARCH Model

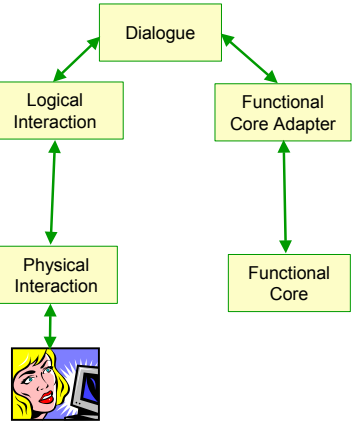


ARCH Model



- Differs from Seeheim
  - *Presentation* split into:
    - Logical Interaction
      - Logical rendering of interaction in terms of mouse events, drawing commands, etc.
    - Physical Interaction
      - Actual rendering of interaction on real device (monitor, keyboard, mouse, …)

ARCH Model



- Differs from Seeheim
  - *Functional Core Adapter* is Application Interface
  - *Functional Core* is Application

Callback Architectures

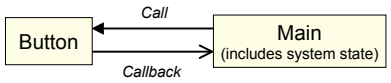
Callback Architectures

- Standard architecture for user interface development on PC, Macintosh, Unix, …
- Calls back to a source component on basis of
  - Events (e.g., a button click, menu item selection, scroll bar click)
  - Property changes (e.g., change of value in text entry box)

Example

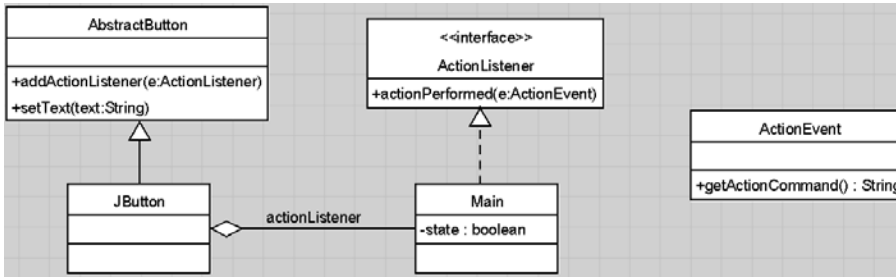
- Implementing a Button
  - Very simple example: an "on/off" button
    - Clicking the button sets the system state to "on"
    - Clicking again sets the system state to "off"

Example



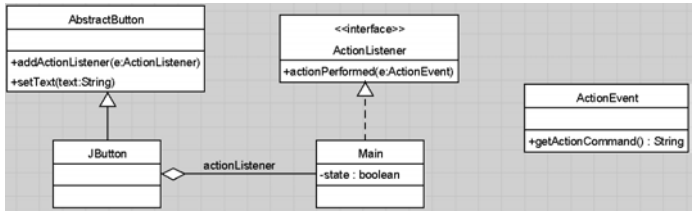
- A *call* is a standard method call
- A *callback* is a call back to some registered component
  - Callbacks must be registered – here *Main* registers itself to be called back whenever button events occur
  - Callbacks are anonymous – the source of the callback does not know what the target is (until the target registers)

Example in Java

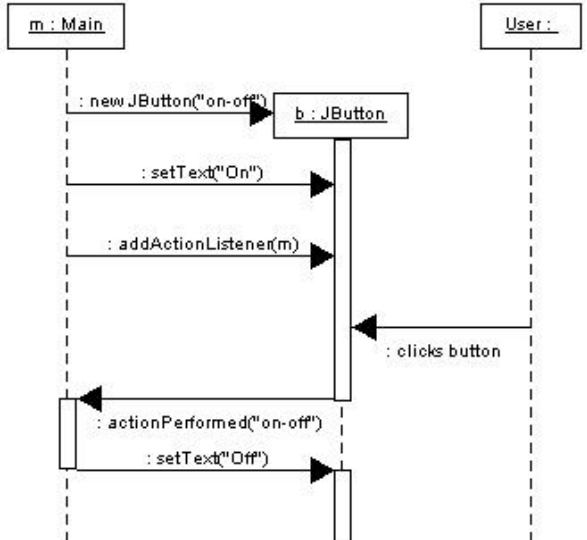


Example in Java

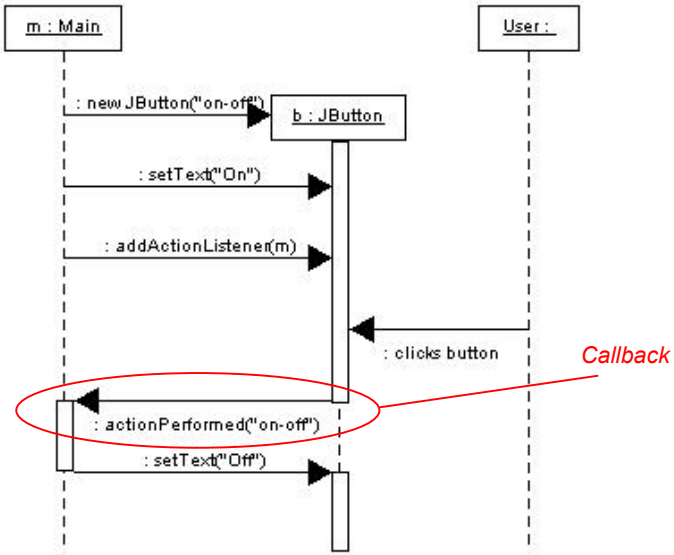
- *Main* registers itself as "listener" to action events from the button
  - *Main* must implement the *ActionListener* interface
- When a user clicks the button, the *JButton* issues an *ActionEvent* to all listeners by calling back to the *actionPerformed* method in the listener (i.e., *Main*)



Sequence of Calls/Callbacks



Sequence of Calls/Callbacks



Simplified Java Code

```
import java.awt.event.*;
import javax.swing.*;

class Main implements ActionListener ()
{
    JButton onOffButton;

    public Main()
    {
        // This is simplified -- actually need to create a frame
        // and put this button in the frame.
        onOffButton = new JButton ("on-off");
        onOffButton.setText ("On");
        onOffButton.addActionListener (this);
    }

    public actionPerformed (e:ActionEvent)
    {
        onOffButton.setText ("Off");
    }
}
```

Discussion

- Relate this architecture to Seeheim

Callbacks: Summary

- Allow reuse of components
  - E.g., JButton can call back to any component
- Leads to "spaghetti" style of programming
  - Callbacks all over obscure structure of program