Comparison of Window Systems

A/Prof. Leow Wee Kheng
CS3249 User Interface Development
Department of Computer Science, National University of Singapore

10 Jan 2014

1 Introduction

Three computing platforms are commonly used in desktop and laptop computers, namely Microsoft Windows, Mac OS/X and Linux. While Microsoft Windows and Mac OS X run their proprietary window systems, Linux computers typically run the X Window System as the low-level window system. On top of X Window, a variety of higher-level frameworks for GUI implementation are available such as Tcl/Tk, FLTK, GTK and Qt. In particular, Qt is popular for its cross-platform capability and elegant design. These window systems and frameworks have their own features, peculiarities, strengths and weaknesses. By comparing their language features and mechanisms, we can better understand their design and implementation.

This document compares the main features between Qt, X Window, Mac Cocoa and Microsoft Foundation Class with respect to the implementation of a simple GUI. Qt implementation is set as the reference because of its simplicity and elegance. Other implementations try to mirror the behaviour of the Qt implementation. In this way, this document may also serve as a bridge for programmers who are familiar with one of the systems to quickly cross over to the other systems.

2 Age Dialog

For concrete comparisons, we use a simple GUI called the age dialog that consists of a spin box and a slider (Fig. 1). A user can interact with the age dialog by typing in the text box of the spin box, click the spin wheel of the spin box, or drag the slider. The spin box and slider are synchronised such that changing one of them causes the other to be updated.

2.1 Qt

In Qt, the UI elements that a user interacts with are called widgets. The base class of all widget subclasses is `QWidget`. Any widget without a parent widget is displayed as a window. So, there is no window class in Qt. The dialog class `QDialog` is a subclass of `QWidget` with additional functions, signals and slots. While `QWidget` is generally used for creating new widgets, both `QWidget` and `QDialog` can be used for creating new dialogs.

There are two ways to create new widgets in Qt:

1. Composition

   A new widget can be created by simply compositing existing widgets together. The widgets are spatially arranged using subclasses of `QLayout`. A main layout manager organises all the underlying

layout managers and widgets, and is set as the layout manager of an empty `QWidget` instance. This `QWidget` instance becomes the parent widget of the underlying widgets, which can be used in yet another widget or displayed as an independent window.

2. Inheritance

   This approach is essentially the same as the composition approach in terms of widget layout, except that a subclass of `QWidget` is defined, whose instance is the parent widget of the underlying widgets.

Therefore, these two approaches for creating widgets are very consistent in Qt.

The age dialog in Qt (Fig. 1) consists of a parent `QWidget`, a `QSlider` and a `QSpinBox`, which integrates a text box and a spin wheel. Turning the spin wheel updates the state of the spin box and displays the updated state in the text box. Conversely, typing in the text box also updates the state of the spin box. The states of the spin box and slider are synchronised through signals and slots mechanism.

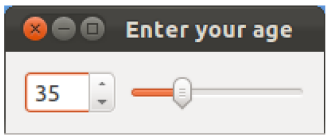


Fig. 1. Qt implementation of the age dialog.

Qt code for the age dialog, taken from [Blan2008], is as follows:

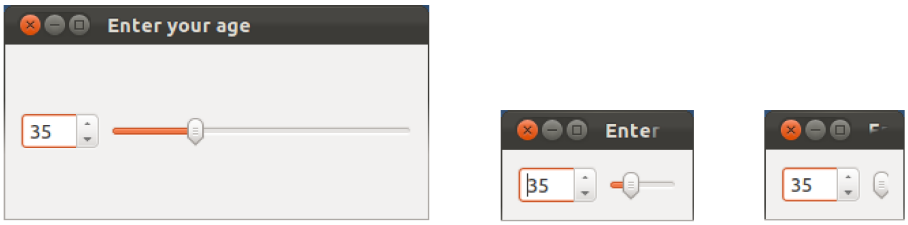
```
// age.cpp
// Qt implementation of the age dialog.

#include <QApplication>
#include <QHBoxLayout>
#include <QSlider>
#include <QSpinBox>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    // Create widgets
    QWidget *window = new QWidget;
    window->setWindowTitle("Enter your age");

    QSpinBox *spinBox = new QSpinBox;
    spinBox->setRange(0, 130);

    QSlider *slider = new QSlider(Qt::Horizontal);
    slider->setRange(0, 130);

    // Connect signals to slots
```

```
    QObject::connect(spinBox, SIGNAL(valueChanged(int)),
        slider, SLOT(setValue(int)));
    QObject::connect(slider, SIGNAL(valueChanged(int)),
        spinBox, SLOT(setValue(int)));

    spinBox->setValue(35);  // Initialise value.

    // Create layout
    QHBoxLayout *layout = new QHBoxLayout;
    layout->addWidget(spinBox);
    layout->addWidget(slider);
    window->setLayout(layout);

    // Start GUI
    window->show();
    return app.exec();
}
```

The `QApplication` class manages the GUI application's control flow and main setting. It contains the event loop where events from the window system and other sources are processed and dispatched. A unique static instance of `QApplication` called `app` is created. The event loop is started by calling `app.exec()` after the widgets are created.

The widgets are created dynamically using the C++ `new` command. Their attributes are set using their respective functions (also called methods). When the widgets are created, they are not visible. After age dialog is created, it is exposed with the widget's `show` function.

Layout of the widget is managed by a `QHBoxLayout` layout manager, which is not a widget. Qt distinguishes between widget and layout manager. Widgets take care of presentation and interaction, whereas layout manager takes care of placement and resizing. `QHBoxLayout` automatically aligns the widgets to the vertical centre. The widgets are set with default resize policy which `QHBoxLayout` tries to respect during resizing. When the age dialog is expanded (Fig. 2a), the widgets remain in the vertical centre. The spin box is not resized. The height of the slider is fixed but its width can change according to the dialog's width. When the dialog is shrunken (Fig. 2b), its size cannot shrink below a minimum size (Fig. 2c), keeping the spin box and the slider in view.



(a) (b) (c)

Fig. 2. Resizing of Qt age dialog. (a) Expansion, (b) shrinking, (c) minimum size.

Qt also provides visual tools for application development: Qt Creator and Qt Designer. Qt Creator is an IDE. Qt Designer allows the user to visually develop a GUI as a widget (`QWidget`), dialog (`QDialog`) or main window (`QMainWindow`). It saves the composition of the GUI in a `.ui` file in XML format, which is easily readable by human and machine. When the program is compiled, a `.h` file of the GUI is automatically generated, which contains the Qt instructions for creating the GUI. Nevertheless, it is just as easy to develop Qt applications using basic editors and debuggers.

Communications between the widgets are achieved with the signals and slots mechanism, a higher-level mechanism than event handling. In particular, the `valueChanged` signals of `spinBox` and `slider` are connected to each other's `setValue` slot (Table 1). Signals and slots mechanism is the most distinguished feature of Qt, and by far the most elegant mechanism of all window systems. The advantages of signals and slots mechanism are discussed in detail in [Blan2008].

Table 1. Qt signal and slot connections.

Source Widget	Signal	Target Widget	Slot
`spinBox`	`valueChanged`	`slider`	`setValue`
`slider`	`valueChanged`	`spinBox`	`setValue`

2.2 X/Motif

X Window was designed and implemented in C, before the invention of C++. The term *widget*, i.e., window gadget, was first used in X Window to refer to a UI element that a user interacts with. Widgets are defined in terms of C `struct`, which does not support inheritance. Thus, composition is the only way to create new widgets. X Window comes with built-in widgets called the athena widgets xaw. Nevertheless, Motif widgets are popularly used with X Window, giving rising to the name X/Motif.

Motif widgets can be created in two ways:

1. Use Xt's generic widget creation function.

   ```
   widget = XtVaCreateManagedWidget(title, widget_class, parent,
       attribute1, value1, ..., NULL)
   ```

   This function creates a managed widget of class `widget_class`. The widget's attribute values can be set along with the widget creation function using pairs of attribute names and values.

2. Use Motif's widget creation function.

   ```
   widget = XmCreateSpinBox(parent, titles, arguments, number_of_arguments)
   ```

   This function creates an unmanaged `SpinBox` widget. After creation, the following function should be called to manage the widget.

   ```
   XtManageChild(widget);
   ```

The age dialog in X/Motif (Fig. 3) consists of a parent widget, a `SpinBox`, a `TextField` and a `Scale` (Motif's slider). Motif has two kinds of spin boxes: `SpinBox` and `SimpleSpinBox`. A `SimpleSpinBox` comes with an accompanying child `TextField`, but there is no obvious way to access the child `TextField` for synchronisation. The `SpinBox` does not have a predefined child `TextField`. One or more child `TextFields` can be added to a `SpinBox`. The age dialog uses a `SpinBox` with a child `TextField`.

Fig. 3. X/Motif implementation of the age dialog.

X/Motif code for the age dialog is as follows:

```
// age.c
// X/Motif implementation of the age dialog.

#include <Xm/Xm.h>
#include <Xm/SpinB.h>
#include <Xm/SSpinB.h>
#include <Xm/TextF.h>
#include <Xm/Scale.h>
#include <Xm/Label.h>
#include <Xm/RowColumn.h>
#include <Xm/Form.h>
#include <string.h>

XtCallbackProc spinTextVerify(Widget, Widget, XmAnyCallbackStruct *);
XtCallbackProc spinBoxValueChanged(Widget, Widget, XmSpinBoxCallbackStruct *);
XtCallbackProc sliderValueChanged(Widget, Widget, XmScaleCallbackStruct *);

#define MinValue 0
#define MaxValue 130
#define InitValue 35


int main (int argc, char *argv[])
{
    XtAppContext app;
    Widget window, layout, spinBox, spinText, slider;

    window = XtVaAppInitialize(&app, "top", NULL, 0,
        &argc, argv, NULL, NULL);

    layout = XtVaCreateManagedWidget("layout",
        xmFormWidgetClass, window, NULL);

    spinBox = XtVaCreateManagedWidget("spin box",
```

```
        xmSpinBoxWidgetClass, layout,
        XmNleftAttachment, XmATTACH_FORM,
        XmNleftOffset, 5, NULL);

    spinText = XtVaCreateManagedWidget("spin box text",
        xmTextFieldWidgetClass, spinBox,
        XmNspinBoxChildType, XmNUMERIC,
        XmNminimumValue, MinValue,
        XmNmaximumValue, MaxValue,
        XmNcolumns, 4,
        XmNwrap, FALSE,
        XmNpositionType, XmPOSITION_VALUE,
        XmNposition, InitValue, NULL);

    slider = XtVaCreateManagedWidget("slider",
        xmScaleWidgetClass, layout,
        XmNwidth, 120,
        XmNorientation, XmHORIZONTAL,
        XmNminimum, MinValue,
        XmNmaximum, MaxValue,
        XmNvalue, InitValue,
        XmNleftAttachment, XmATTACH_WIDGET,
        XmNleftWidget, spinBox,
        XmNleftOffset, 5,
        XmNrightAttachment, XmATTACH_FORM,
        XmNrightOffset, 5,
        XmNtopAttachment, XmATTACH_FORM,
        XmNtopOffset, 8, NULL);

    XtAddCallback(spinText, XmNactivateCallback, spinTextVerify, slider);
    XtAddCallback(spinBox, XmNvalueChangedCallback, spinBoxValueChanged, slider);
    XtAddCallback(slider, XmNdragCallback, sliderValueChanged, spinText);
    XtAddCallback(slider, XmNvalueChangedCallback, sliderValueChanged, spinText);

    XtRealizeWidget(window);
    XtAppMainLoop(app);
    return 0;
}


XtCallbackProc spinTextVerify(Widget spinText, Widget slider,
    XmAnyCallbackStruct *cbs)
{
    int ret, value, changed;
    char *str, string[100];

    changed = 0;
    str = XmTextFieldGetString (spinText);
    if (str && (ret = sscanf(str, "%d", &value)) > 0)
    {
```

```
        sprintf(string, "%d", value);
        if (strlen(string) == strlen(str) && value >= MinValue && value <= MaxValue)
        {
            XtVaSetValues(spinText, XmNposition, value, 0);
            XtVaSetValues(slider, XmNvalue, value, 0);
            changed = 1;
        }
    }

    if (str)
        XtFree(str);

    if (!changed)
    {
        XtVaGetValues(spinText, XmNposition, &value, 0);
        sprintf(string, "%d", value);
        XmTextFieldSetString(spinText, string);  // Reset text field.
    }
}


XtCallbackProc spinBoxValueChanged(Widget spinBox, Widget slider,
    XmSpinBoxCallbackStruct *cbs)
{
    int value;

    XtVaGetValues(cbs->widget, XmNposition, &value, 0);
    XtVaSetValues(slider, XmNvalue, value, 0);
}


XtCallbackProc sliderValueChanged(Widget slider, Widget spinText,
    XmScaleCallbackStruct *cbs)
{
    int value;

    XtVaGetValues(slider, XmNvalue, &value, 0);
    XtVaSetValues(spinText, XmNposition, value, 0);
}
```

X's equivalent of `QApplication` is `XtAppContext`. An instance of `XtAppContext` called `app` is created statically. The event loop is started with the function `XtAppMainLoop(app)`.

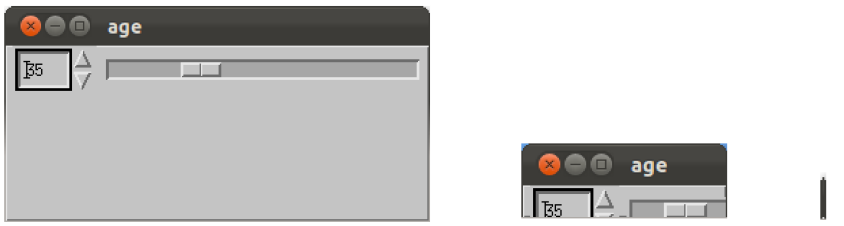
The dialog window is created with `XtVaAppInitialize`, whereas the other widgets are created with `XtVaCreateManagedWidget`. Their attributes are specified as arguments to the widget creation functions. Te widgets are created invisible. After the whole dialog is created, it is exposed by the function `XtRealizeWidget`.

Layout of the widgets is managed by the layout manager `layout`, of type `xmFormWidgetClass`. Unlike

Qt layout manager, X/Motif layout manager is also a widget. Placements of the widgets are specified as arguments to the widget creation functions. In contrast, placement of a Qt widget is specified when adding the widget to the layout manager using the `addWidget` function.

Motif has a `RowColumn` layout manager that is similar to Qt's `QBoxLayout`. However, `RowColumn` cannot centre a slider vertically; vertical positioning works only for labels and texts. So, the `Form` widget is used instead. `Form` widget uses an attachment scheme to specify the placements and resizing of widgets. By default, a widget is placed at the top-left corner of its parent `Form`. It can be specified to attach to the left, right, top or bottom sides of the `Form` or other widgets in the `Form`. Attaching to the left or right side allows a widget to still slide up and down as the window resizes. Similarly, attaching to the top or bottom side allows a widget to slide left and right in the window. To fix a widget's location, it has to be attached on two orthogonal sides. The sample program attaches `spinBox`, together with its child `spinText`, to the left of `layout`. which is a `Form` widget, and attaches `slider` to the left of `spinBox` and right of `layout`. In addition, `slider` is attached to the top of `layout` with an offset so that `slider` is vertically aligned to the centre of `spinBox` (Fig. 3).

When the age dialog is expanded, `spinBox`'s size remains unchanged while `slider` is stretched horizontally (Fig. 4a). Both widgets remain fixed to the top of the dialog. By default, the age dialog can be shrunken so much the widgets can be partly hidden (Fig. 4b) or totally hidden (Fig. 4c).



(a) (b) (c)

Fig. 4. Resizing of X/Motif age dialog. (a) Expansion, (b) shrinking, (c) Minimum size.

The widgets are responsible only for displaying window content. Their behaviours are achieved by callback functions. Each callback is associated to a widget and an event using `XtAddCallback` (Table 2). Auxiliary data can be passed to `XtAddCallback`. When the user interacts with a widget, an event is added to the event queue. The event dispatcher running the event loop removes an event from the event queue and calls the associated callback, with the associated widget and data as the arguments.

The callback `spinTextVerify` is called when the user activates the text box (`XmNactivateCallback` event) by pressing the enter key in the text box. It verifies that the text maintained in `spinText` is a valid integer value. It updates the values of `slider` and `spinText`, which maintains the value of its parent `spinBox`. The callback `spinBoxValueChanged` is called when the user clicks the spin wheel up or down, which causes its value to change (`XmNvalueChangedCallback` event). The value of its child `spinText` is updated automatically, which is used by the callback to update the value of `slider`. The callback `sliderValueChanged` is called when the `slider` is dragged (`XmNdragCallback` event) or when the user releases the mouse button after dragging, which causes the value to change (`XmNvalueChangedCallback` event). It gets the value of `slider` and updates the value of `spinText`.

Table 2. X/Motif callback table.

Widget	Event	Callback	Data
spinText	XmNactivateCallback	spinTextVerify	slider
spinBox	XmNvalueChangedCallback	spinBoxValueChanged	slider
slider	XmNdragCallback	sliderValueChanged	spinText
slider	XmNvalueChangedCallback	sliderValueChanged	spinText

Some notes on program syntax: If you get the following warning messages from your compiler, it is safe to ignore them and your program still runs properly. However, if you really want to turn off the warning messages, then do the following:

- missing sentinel in function call

  This is a result of using `0` instead of `NULL` to end the argument lists of `XtVaGetValues` and `XtVaSetValues`. To fix it, just change `0` to `NULL`.

- 'XtAddCallback' from incompatible pointer type

  This results from declaring the prototype of the callbacks as

  ```
  XtCallbackProc callback(Widget, Widget, XmAnyCallbackStruct *);
  ```

  or other forms. Changing the prototype to

  ```
  void callback(Widget, XtPointer, XtPointer);
  ```

  fixes the problem. In this case, you can cast the 2nd and 3rd arguments to the proper types in the callback function as follows:

  ```
  void sliderValueChanged(Widget slider, XtPointer wp, XtPointer cp)
  {
      int value;

      Widget textField = (Widget) wp;
      XmScaleCallbackStruct *cbs = (XmScaleCallbackStruct *) cp;

      XtVaGetValues(slider, XmNvalue, &value, NULL);
      XtVaSetValues(textField, XmNposition, value, NULL);
  }
  ```

Like Qt codes, X/Motif codes are also straightforward and easy to understand. Motif widgets are responsible only for handling graphic appearance on the screen. Their behaviours are programmed in the callback functions. In comparison, Qt widgets are functionality complete, which can be used as they are by connecting signals to appropriate slots. The similarity in the programming style and naming of the widgets, events and functions between Qt and X/Motif suggests that the creators of Qt, Haavard Nord and Eirik Chambe-Eng, adopted some good practices of X/Motif in Qt, including the term *widget*.

2.3 Mac Cocoa

Mac OS X provides the Cocoa framework for building GUI. Cocoa differentiates between window and view. An `NSWindow` object manages a physical window on the screen. An object of `NSView` class or its subclass owns a rectangular region associated with a window. It produces the image content for the region and responds to events occurring in it. The content view is the top-level view of a window. It contains the view for each UI element such as text field view, button view and slider view. Cocoa also have panel, a `NSPanel` instance, which is a special kind of window analogous to `QDialog` in Qt.

In Cocoa, the UI elements that a user interacts with are called controls, which are instances of `NSView` subclasses. A control can contain one ore more cells (Fig. 5), which are rectangular areas within a control and they are instances of `NSCell` subclasses. Control is responsible for displaying the GUI component, accepting user events and sending messages (i.e., events) to other objects, though it usually delegates the first two responsibilities to the cells, so that text or images can be displayed in a view without the full overhead of an `NSView` subclass.

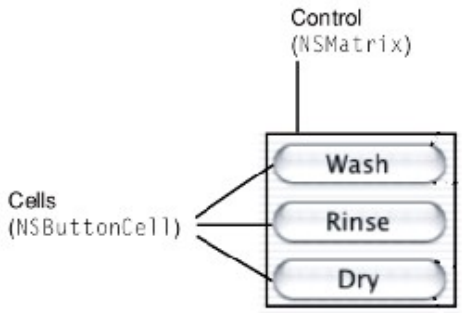


Fig. 5. In Cocoa, a UI element is called a control and it can contain multiple cells.

GUI is developed using Xcode, Mac's IDE and GUI builder. UI elements are selected and placed in Xcode, which generates a specification of the GUI in a nib file (“nib” stands for “NeXT Interface Builder”). Similar to Motif implementation, Cocoa implementation of the age dialog (Fig. 6) uses three UI elements: text field, spin wheel (called stepper in Cocoa) and slider. Unlike Motif, however, the spin wheel and the text field are separate controls that maintain their own states.

Fig. 6. Cocoa implementation of age dialog.

The main Cocoa code written in Objective-C is as follows:

```
//  main.m

#import <Cocoa/Cocoa.h>

int main(int argc, char *argv[])
{
    return NSApplicationMain(argc, (const char **)argv);
}


//  AgeDialog.h

#import <Cocoa/Cocoa.h>

@interface AgeDialog : NSObject <NSApplicationDelegate>

@property (assign) IBOutlet NSWindow *window;
@property (assign) IBOutlet NSTextField *spinText;
@property (assign) IBOutlet NSStepper *spinWheel;
@property (assign) IBOutlet NSSlider *slider;

@end


// AgeDialog.m

#import "AgeDialog.h"

#define MIN_AGE 0
#define MAX_AGE 130

@implementation AgeDialog

- (void)dealloc
{
    [super dealloc];
}


- (void)applicationDidFinishLaunching:(NSNotification *)aNotification
{
    // Insert code here to initialize your application
    [[self spinText] setIntegerValue:35];
    [[self spinWheel] setIntegerValue:[[self spinText] integerValue]];
    [[self slider] setIntegerValue:[[self spinText] integerValue]];
}
```

```
- (IBAction)textChanged:(NSTextField *)sender
{
    // Use scanner class to parse string into integer
    NSScanner* scanner = [NSScanner scannerWithString:[sender stringValue]];
    NSInteger result;
    bool status = [scanner scanInteger:&result];

    if(status == YES && scanner.scanLocation == [[sender stringValue] length]
       && [sender integerValue] >= MIN_AGE && [sender integerValue] <= MAX_AGE)
    {
        [[self slider] setIntegerValue:[sender integerValue]];
        [[self spinWheel] setIntegerValue:[sender integerValue]];
    }
    else
    {
        // Otherwise reset to original value as that of slider
        [[self spinText] setIntegerValue:[[self slider] integerValue]];
    }
}


- (IBAction)spinWheelValueChanged:(NSStepper *)sender {
    [[self spinText] setIntegerValue:[sender integerValue]];
    [[self slider] setIntegerValue:[sender integerValue]];
}


- (IBAction)sliderValueChanged:(NSSlider *)sender {
    [[self spinText] setIntegerValue:[sender integerValue]];
    [[self spinWheel] setIntegerValue:[sender integerValue]];
}

@end
```

Cocoa's equivalent of `QApplication` in `NSApplication`. An instance of `NSApplication` called `NSApp` is automatically created which maintains the event loop. The method `NSApplicationMain` creates the objects specified in the nib file and wake them up. Then, the method `applicationDidFinishLaunching` is invoked, which initialises the window and controls. Finally, the event loop is run.

The `AgeDialog` is defined as a subclass of `NSObject`, the parent class of all Objective-C classes, using the `@interface` keyword, and it implements the protocol of `NSApplicationDelegate`. Objective-C protocol is similar to Java interface that specifies the methods that an object class implements. The keyword `IBOutlet` defines a special instance variable that points to another object of a particular class (Fig. 7). `AgeDialog` has four outlets, namely `window`, `spinText`, `spinWheel` and `slider`, each pointing to an object of a different class, respectively, `NSWindow`, `NSTextField`, `NSStepper` and `NSSlider`. Note that `AgeDialog` itself is not a window. Instead, it has an outlet that points to an `NSWindow` object. The `@property` keyword declares that the pair of methods known as a setter and a getter for the instance variables. For a variable named `value`, the setter method is named `setValue` and the getter method is named `value`. For `AgeDialog`, the getters and setters simply get and set the references (or pointers) to the controls.

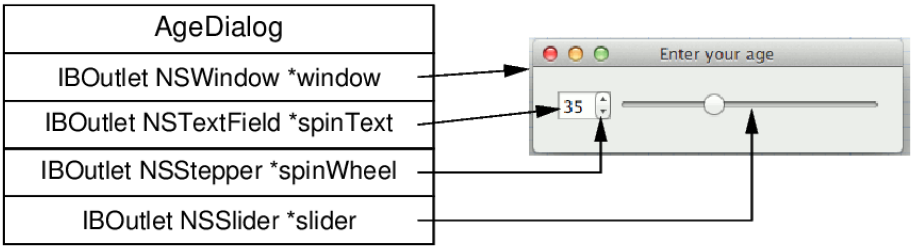


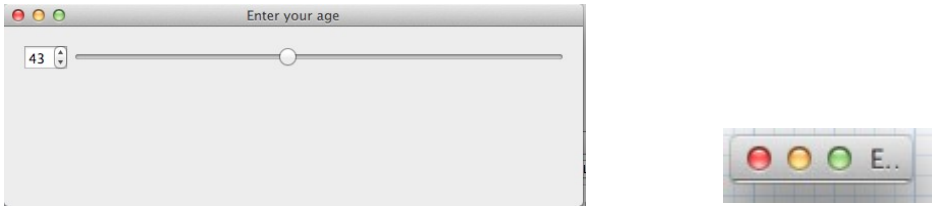
Fig. 7. `AgeDialog` is an object that encapsulates the GUI. It contains outlets that point to the window and controls.

The range of values of the controls can be specified in Xcode or written explicitly in `AgeDialog.m`. In this example, they are encoded in the nib file. Similarly, the initial value of 35 can be specified in either way. In this example, it is written in the function `applicationDidFinishLaunching`.

Layout of the controls are specified in Xcode, and encoded in the nib file. Resizing of the age dialog works in a similar way as does X/Motif. When the dialog is expanded, the slider is stretched horizontally and the spin box remains unchanged (Fig. 8a). The dialog can be shrunken until the controls are not visible (Fig. 8b).

Like X/Motif, Cocoa controls are responsible only for displaying window content. Their behaviours are achieved by Cocoa's target-action mechanism (Fig. 9). Actions are special methods in target objects indicated with the `IBAction` keyword. They are triggered when their associated controls are activated by the user. `AgeDialog` has three action methods. Connections between controls and their targets and actions are specified in Xcode. There is no need to explicitly refer to events in the source code of `AgeDialog`.

The method `textChanged` is called when the user presses enter key after changing the text box. It scans for an integer in the text string entered and checks that the integer value is within range. Then, it calls `spinWheel` and `slider` to update their values. The method `spinWheelValueChanged` is called when `spinWheel` is clicked up or down, and it calls `slider` and `spinText` to update their values. Similarly, the method `sliderValueChanged` is called when `slider` is dragged, and it calls `spinWheel` and `spinText` to update their values. Thus, Cocoa's target-action mechanism is analogous to X/Motif's callback mechanism.



(a) (b)

Fig. 8. Resizing of Cocoa age dialog. (a) Expansion, (b) minimum size.

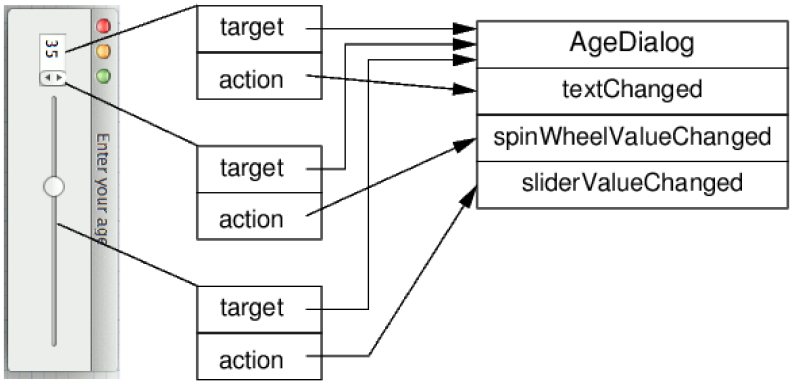


Fig. 9. Cocoa's target-action mechanism associates GUI elements with their target objects and action methods.

Except for the specifications that are hidden in the nib file, which includes creation, layout and initialisation of control objects and connections between control objects and their targets and actions, the action methods are relatively straightforward and easy to understand.

2.4 MFC

Microsoft Foundation Class (MFC) is a higher-level framework on top of Windows API for developing GUI applications. MFC provides two main types of top-level window, namely frame window `CFrameWnd` and dialog `CDialog`, both are subclasses of `CWnd`. `CFrameWnd` and `CDialog` are analogous to Qt's `QMainWindow` and `QDialog`.

In MFC, the UI element that a user interacts with are called controls, as for Cocoa. Although windows, dialogs and controls are all subclasses of `CWnd`, their behaviours are not as uniform as those of `QWidget` subclasses. MFC differentiates between frame window class `CFrameWnd`, dialog class `CDialog` and controls. `CFrameWnd` is analogous to `QMainWindow` whereas `CDialog` is analogous to `QDialog`.

GUI is developed using an IDE such as Visual Studio. UI elements are selected and placed in Visual Studio. Similar to Motif implementation, MFC implementation of the age dialog (Fig. 10) uses three UI elements, namely `CEdit`, `CSpinButtonCtrl` and `CSliderCtrl`. The text field and the spin wheel are linked in Visual Studio such that their states are synchronised.

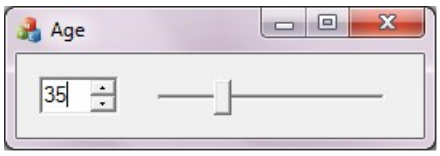


Fig. 10. MFC implementation of age dialog.

Visual Studio generates as many as 15 files and several subdirectories for the age dialog. Most of the files do not need to be changed. The four most relevant files are listed below, two for the application and two for the dialog. The black and blue parts are automatically generated by Visual Studio, whereas the red parts are manually added either in Visual Studio or directly in the source files.

```
//-----------------------------------------------------------------
// Age.h : main header file for the PROJECT_NAME application

#pragma once

#ifndef __AFXWIN_H__
    #error "include 'stdafx.h' before including this file for PCH"
#endif

#include "resource.h"           // main symbols

// AgeApp:
// See Age.cpp for the implementation of this class

class AgeApp : public CWinApp
{
public:
    AgeApp();

// Overrides
public:
    virtual BOOL InitInstance();

// Implementation

    DECLARE_MESSAGE_MAP()
};

extern AgeApp theApp;


//-----------------------------------------------------------------
// Age.cpp : Defines the class behaviors for the application.

#include "stdafx.h"
#include "Age.h"
#include "AgeDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// AgeApp
```

```
BEGIN_MESSAGE_MAP(AgeApp, CWinApp)
   ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()


// AgeApp construction

AgeApp::AgeApp()
{
   // TODO: add construction code here,
   // Place all significant initialization in InitInstance
}


// The one and only AgeApp object

AgeApp theApp;


// AgeApp initialization

BOOL AgeApp::InitInstance()
{
   CWinApp::InitInstance();

   // Create the shell manager, in case the dialog contains
   // any shell tree view or shell list view controls.
   CShellManager *pShellManager = new CShellManager;

   // Activate "Windows Native" visual manager for enabling themes in MFC controls
   CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerWindows));

   // Standard initialization
   // If you are not using these features and wish to reduce the size
   // of your final executable, you should remove from the following
   // the specific initialization routines you do not need
   // Change the registry key under which our settings are stored
   // TODO: You should modify this string to be something appropriate
   // such as the name of your company or organization
   SetRegistryKey(_T("Local AppWizard-Generated Applications"));

   AgeDialog dlg;
   m_pMainWnd = &dlg;
   INT_PTR nResponse = dlg.DoModal();
   if (nResponse == IDOK)
   {
      // TODO: Place code here to handle when the dialog is
      //  dismissed with OK
   }
   else if (nResponse == IDCANCEL)
```

```
    {
        // TODO: Place code here to handle when the dialog is
        //  dismissed with Cancel
    }
    else if (nResponse == -1)
    {
        TRACE(traceAppMsg, 0, "Warning: dialog creation failed, so application is
terminating unexpectedly.\n");
        TRACE(traceAppMsg, 0, "Warning: if you are using MFC controls on the dialog, you
cannot #define _AFX_NO_MFC_CONTROLS_IN_DIALOGS.\n");
    }

    // Delete the shell manager created above.
    if (pShellManager != NULL)
    {
        delete pShellManager;
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    //  application, rather than start the application's message pump.
    return FALSE;
}


//-----------------------------------------------------------------
// AgeDialog.h : header file

#pragma once

#include "afxcmn.h"
#include "afxwin.h"

// AgeDialog dialog
class AgeDialog : public CDialog
{
// Construction
public:
    AgeDialog(CWnd* pParent = NULL);    // standard constructor

// Dialog Data
    enum { IDD = IDD_AGE_DIALOG };

    protected:
    virtual void DoDataExchange(CDataExchange* pDX);// DDX/DDV support

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
```

```
    virtual BOOL OnInitDialog();
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnEnUpdateSpintext();
    afx_msg void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    DECLARE_MESSAGE_MAP()

private:
    int age;
    CEdit spinText;
    CSpinButtonCtrl spinWheel;
    CSliderCtrl slider;
};


//-----------------------------------------------------------------
// AgeDialog.cpp : implementation file
//

#include "stdafx.h"
#include "Age.h"
#include "AgeDialog.h"
#include "afxdialogex.h"

// AgeDialog dialog

AgeDialog::AgeDialog(CWnd* pParent /*=NULL*/)
    : CDialog(AgeDialog::IDD, pParent)
    , age(0)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void AgeDialog::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_SPINTEXT, spinText);
    DDX_Control(pDX, IDC_SPINWHEEL, spinWheel);
    DDX_Control(pDX, IDC_SLIDER, slider);
}

BEGIN_MESSAGE_MAP(AgeDialog, CDialog)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_EN_UPDATE(IDC_SPINTEXT, &AgeDialog::OnEnUpdateSpintext)
    ON_WM_HSCROLL()
END_MESSAGE_MAP()


// AgeDialog message handlers
```

```
BOOL AgeDialog::OnInitDialog()
{
   CDialog::OnInitDialog();

   // Set the icon for this dialog.  The framework does this automatically
   //  when the application's main window is not a dialog
   SetIcon(m_hIcon, TRUE);            // Set big icon
   SetIcon(m_hIcon, FALSE);           // Set small icon

   // TODO: Add extra initialization here
   spinWheel.SetRange(0, 130);
   slider.SetRange(0, 130);
   spinWheel.SetPos(35);

   return TRUE;  // return TRUE  unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void AgeDialog::OnPaint()
{
   if (IsIconic())
   {
      CPaintDC dc(this); // device context for painting

      SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

      // Center icon in client rectangle
      int cxIcon = GetSystemMetrics(SM_CXICON);
      int cyIcon = GetSystemMetrics(SM_CYICON);
      CRect rect;
      GetClientRect(&rect);
      int x = (rect.Width() - cxIcon + 1) / 2;
      int y = (rect.Height() - cyIcon + 1) / 2;

      // Draw the icon
      dc.DrawIcon(x, y, m_hIcon);
   }
   else
   {
      CDialog::OnPaint();
   }
}


// The system calls this function to obtain the cursor to display while the user drags
//  the minimized window.
```

```
HCURSOR AgeDialog::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}


void AgeDialog::OnEnUpdateSpintext()
{
    // TODO:  If this is a RICHEDIT control, the control will not
    // send this notification unless you override the CDialog::OnInitDialog()
    // function to send the EM_SETEVENTMASK message to the control
    // with the ENM_UPDATE flag ORed into the lParam mask.

   int value = GetDlgItemInt(IDC_SPINTEXT);

    if (value > 130 || value < 0)
    {
       value = age;
       SetDlgItemInt(IDC_SPINTEXT, age);
    }

    age = value;
    if (slider)
         slider.SetPos(age);
}


void AgeDialog::OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    // TODO: Add your message handler code here and/or call default

    if(pScrollBar == (CScrollBar *) &slider)
    {
       int value = slider.GetPos();
       age = value;
       SetDlgItemInt(IDC_SPINTEXT, age);
    }
    else
       CDialog::OnHScroll(nSBCode, nPos, pScrollBar);

    CDialog::OnHScroll(nSBCode, nPos, pScrollBar);
}
```

MFC's equivalent of `QApplication` is `CWndApp`. Unlike Qt, X and Cocoa which simply use an instance of their application classes, MFC requires the creation of `AgeApp` as a subclass of `CWndApp`, even though usually nothing needs to be changed in `AgeApp`. An unique instance of `AgeApp` called `theApp` is created as a global static object in Age.cpp. The C++ `main` function is automatically provided by MFC, and `CWndApp::Run()` is called automatically to run the event loop.

Examination of Age.h and Age.cpp shows that these files contain rather complicated instructions. So, the instructions are generated automatically in case the programmer needs to change `AgeApp`. Providing placeholders in the function `InitInstance` to handle different ways by which the `dialog` closes is useful but breaks the coherence of `AgeApp` class and incurs a tighter coupling between `AgeApp` and `AgeDialog`. Automatic generation of several other .h and .cpp files further complicates the implementation. In comparison, the implementations of Qt, X/Motif and Cocoa are cleaner, more coherent and less tightly coupled.

The `AgeDialog` is defined as a subclass of `CDialog`. It contains three controls, namely `spinText`, `spinWheel` and `slider`, which is an instance of `CEdit`, `CSpinButtonCtrl` and `CSliderCtrl`, respectively. Each control is associated with a GUI item identified by an ID number, which are labelled as `IDC_SPINTEXT`, `IDC_SPINWHEEL` and `IDC_SLIDER` (Fig. 11). Synchronisation between the states of the GUI items and the control objects is achieved by the Dialog Data Exchange (DDX) mechanism specified in the function `DoDataExchange`. In contrast, in Qt, X/Motif and Cocoa, the widgets or controls are the actual GUI elements. Cocoa uses outlets to point to the controls and cells, whereas Qt and X/Motif use C/C++ pointers to point to the widgets.

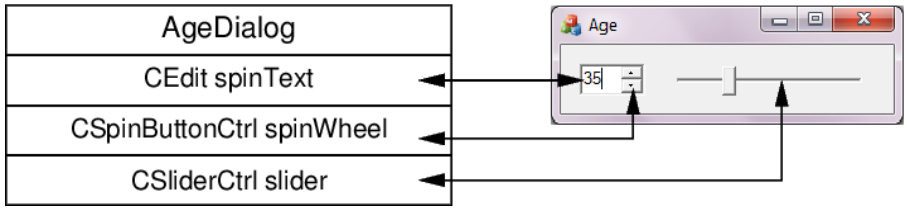


Fig. 11. Dialog data exchange between control objects in `AgeDialog` and GUI items.

Layout of the widow components is specified in Visual Studio, and encoded in the generated files. By default, resizing the window does not affect the size of the controls (Fig. 12). The dialog can be shrunken until the controls are partially or totally invisible (Fig. 12b, c).

(a) (b) (c)

Fig. 12. Resizing of MFC age dialog. (a) Expansion, (b) shrinking, (c) minimum size.

Like X/Motif and Cocoa, MFC GUI items and control objects are responsible only for displaying window content. Their behaviours are achieved by message handlers, MFC's term for event handlers. The events or messages to be handled by the `AgeDialog` are specified in the message map section of `AgeDialog.cpp`. Each event is handled by a prescribed function (Table 3) that `AgeDialog` can override to implement the require behaviour. The WM messages are mapped to default message handlers, which need

not be specified in the message map. On the hand other, the EN message is mapped to a programmer-specified function `OnEnUpdateSpintext` .

Table 3. Message map indicates the events to be processed by `AgeDialog`.

Message	Message Handler
`ON_WM_PAINT`	`OnPaint`
`ON_WM_QUERYDRAGICON`	`OnQueryDragIcon`
`ON_EN_UPDATE`	`OnEnUpdateSpintext`
`ON_WM_HSCROLL`	`OnHScroll`

The method `OnEnUpdateSpintext` is called when the user presses enter key after changing the text box (`ON_EN_UPDATE` event). It obtains the integer value of the text box through its ID `IDC_SPINTEXT`, checks that the value is within range, and set the value of the text box through `IDC_SPINTEXT` and the value of the slider by calling the `SetPos` function of `slider`. The spin button and the text box are linked in Visual Studio, so there is no need to explicitly set the value of the spin button. The method `OnHScroll` is called when the user slides the slider (`ON_WM_HSCROLL` event). It checks whether the control that activated the event is `slider`, and then gets the value of `slider` to update the text box.

The codes for two other message handlers (blue parts), namely `OnPaint` and `OnQueryDragIcon`, are automatically generated. There is no need to change these methods. In fact, they can be completely commented out, i.e., not overridden in `AgeDialog`, and the corresponding methods of the parent class `CDialog` work just as well.

3 Summary

Table 4 summarises the major features of Qt, X Window, Cocoa and MFC. In developing a GUI application, both Qt and X require only the declaration of a static application object. Cocoa requires only the calling of the class function `NSApplicationMain` of `NSApplication`. In contrast, MFC require the creation of a subclass of `CWinApp`. Even though the source code of the application subclass can be generated automatically, the need to create an application subclass indicates that MFC is less elegantly designed as the other frameworks.

In Qt, any widget without a parent is a window. In X, the main window is a widget created by the function `XtVaAppInitialize`. Cocoa and MFC, on the other hand, differentiate between window and UI elements called controls. Qt differentiates between layout manager and widget whereas X layout manager is also a widget. Window layout for Cocoa and MFC are specified in the Xcode and Visual Studio respectively and encoded in project files.

MFC differentiates between GUI items and control objects. GUI items are identified by ID numbers whereas control objects are instances of control classes. Dialog Data Exchange (DDX) mechanism is used to synchronise the states of the GUI items and the control objects. In contrast, Cocoa, X and Qt directly place controls or widgets in the GUI. Cocoa uses outlets to point to window and controls, whereas X and Qt use C/C++ pointers to point to widgets.

Table 4. Comparison of major features of various frameworks.

	Qt	**X/Motif**	**Cocoa**	**MFC**
application	`QApplication` instance	`XtAppContext` instance	`NSApplication` instance	`CWinApp` subclass instance
event loop	`QApplication::exec()`	`XtAppMainLoop()`	in `NSApplication`	`CWinApp::Run()`
top-level window	widget without parent	widget created by `XtVaAppInitialize`	`NSWindow`	`CWnd` subclass
dialog	`QDialog`	`Widget`	`NSPanel`	`CDialog`
UI element	`QWidget` or subclass	`Widget`	control, `NSView` subclass	control, `CWnd` subclass
layout	`QLayout` subclass	layout widget	specified in XCode	specified in Visual Studio
widget reference	C++ pointer	C pointer	`IBOutlet`	Dialog Data Exchange
widget communication	signals and slots	event callback	target-action (similar to event callback)	message map (similar to event callback)
widget behaviour	widget functions	top-level window's callbacks	target top-level window's callbacks	top-level window's message handlers
coding style	short and elegant	long but simple	short and simple	long and complex
code readability	easy to understand, transparent	easy to understand, transparent	easy to understand, some parts hidden	difficult to understand, some parts hidden

Widgets and controls in X Window, Cocoa and MFC are responsible only for displaying window content. Their behaviours are implemented in functions. When an event is dispatched, X Window invokes a callback function that is implemented in the top-level window. Cocoa invokes an action method implemented in target object that serves as the top-level window, whereas MFC invokes a message handler implemented in the top-level window. In contrast, Qt widgets are functionally more complete and can be used as they are in the GUI, and they communicate through signals and slots.

In summary, Qt codes are short, elegant, transparent and easy to understand. X Window codes are longer but simple, transparent and easy to understand. Cocoa codes are short and relatively simple and easy to understand. It is quite transparent except for the parts that are hidden in the nib file. MFC codes are long, complex, often redundant and relatively difficult to understand. Some parts of the code are hidden in various binary coded files.

4 References

1. J. Blanchette and M. Summerfield. *C++ GUI Programming with Qt 4*, 2nd ed., Prentice Hall, 2008.
2. P. Buttfield-Addison and J. Manning. *Learning Cocoa with Objective-C*, 3rd ed., O'Reilly, 2013.
3. *Cocoa Fundamentals Guide*, Apple Computer Inc., 2010.
4. J. D. Davidson and Apple Computer, Inc. *Learning Cocoa with Objective-C*, 2nd ed., O'Reilly, 2002.
5. A. Fountain, J. Huxtable, P. Ferguson and D. Heller. *Motif Programming Manual*, O'Reilly, 2001.
6. I. Horton. *Beginning Visual C++ 2010*, Wiley, 2010.
7. B. J. Keller. *A Practical Guide to X Window Programming*, CRC, 1990.
8. M. Trent and D. McCormack. *Beginning Mac OS X Snow Leopard Programming*, Wiley, 2010.