Views – An Independent GUI Development Tool for Rotor

R. Nigel Horspool, University of Victoria, CA
Judith Bishop, University of Pretoria, ZA

GUI (Graphical User Interfaces):

- Everyone expects programs to use GUIs.

- What is available for C# programs?
  - If the program is to be run on a Windows system, the program can use the Form class in the System.Windows.Forms namespace plus the classes for the various controls.
    (The WinForms solution.)

  - If the program is to be run on Rotor, the portable approach seems to be calls to Tcl using the API provided by the SharedSourceCLI.TK namespace on Rotor.

The WinForms Solution

Coding a Windows form by hand is hard –

- Too many control classes, properties and methods to remember easily,
- Visual layout requires much trial and error.

Sample code for a form with just a single checkbox control appears on the next slide ...

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

public class MyForm : Form {
  private CheckBox cb;

  private void InitializeComponent() {
    cb = new CheckBox();
    SuspendLayout();
    cb.Location = new Point(24, 168);
    cb.Name="checkBox1";  cb.TabIndex=1;  cb.Text="Click Me";
    cb.CheckedChanged += new System.EventHandler(cbClicked);
    Controls.AddRange(new Control[] { cb });
    this.Name = "MyForm";  this.Text = "MyForm";
    ResumeLayout(false);
  }
  private void cbClicked(object sender, EventArgs e) {
    // check box code
  }
}
```

Creating the WinForms Code

- Usually, the code is automatically generated using the MDE (Microsoft Development Environment) in Visual Studio.
- Not available for Rotor.
- Need to code calls to Tcl by hand.
- Sample code appears on next slide.

```
// Sample Tcl/TK Usage

  void InitProc( TclInterp interp ) {
    // Initialize Tcl
    interp.TclInit();
    // Initialize the TK libs
    interp.TkInit();

    interp.Eval("canvas .c");
    interp.Eval(
      "button .c.myButton -text \"Click Me\" -command {myButton};");
    interp.CreateCommand("myButton",
       new TclCmdProc(this.ButtonClickFunction));

    interp.Eval("pack .c");
  }

  // Create the interpreter
  TclInterp m_interp = new TclInterp();

  // and start the Tcl application
  m_interp.TkMain( new TclAppInitProc(InitProc) );
```

Desiderata

- Neither the WinForms code nor the Tcl invocation code should be programmed afresh for each application.
- Consistency across the different platforms would be nice.
- Easy development of simple GUIs without a visual tool would be nice.

... and that is where Views comes in.

VIEWS = *Vendor Independent Event and Windowing System*

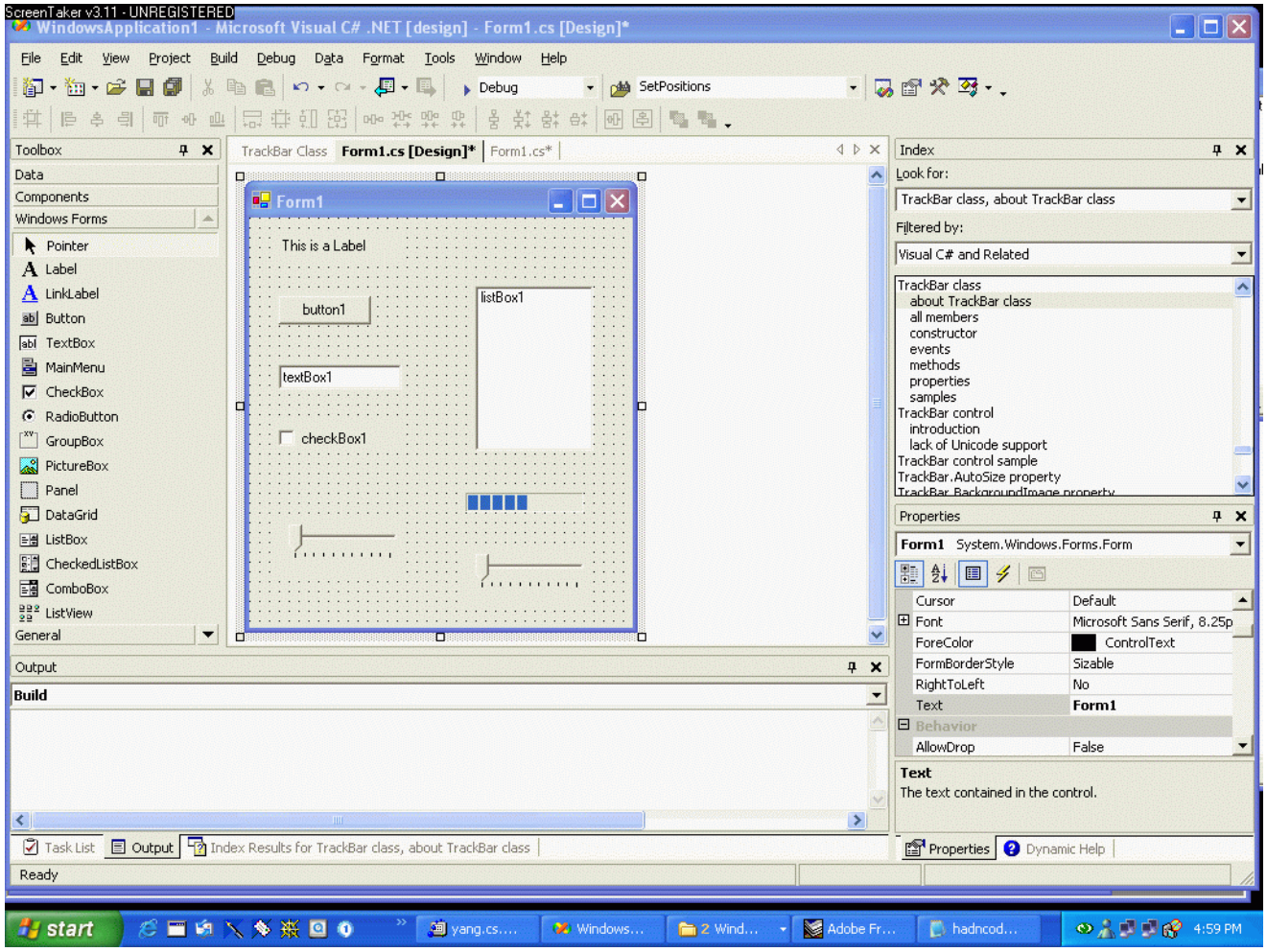
Shall we use a tool to build the GUI?

The programmer has to learn ..

- how to use a development environment,
- how to use the form designer in that environment,
- how to edit the generated code,
- and still has to learn about many classes, and exceptions and ...

The development environment itself looks very complicated and intimidating to all but expert programmers ...

A Snapshot of the MDE



A new approach – Views

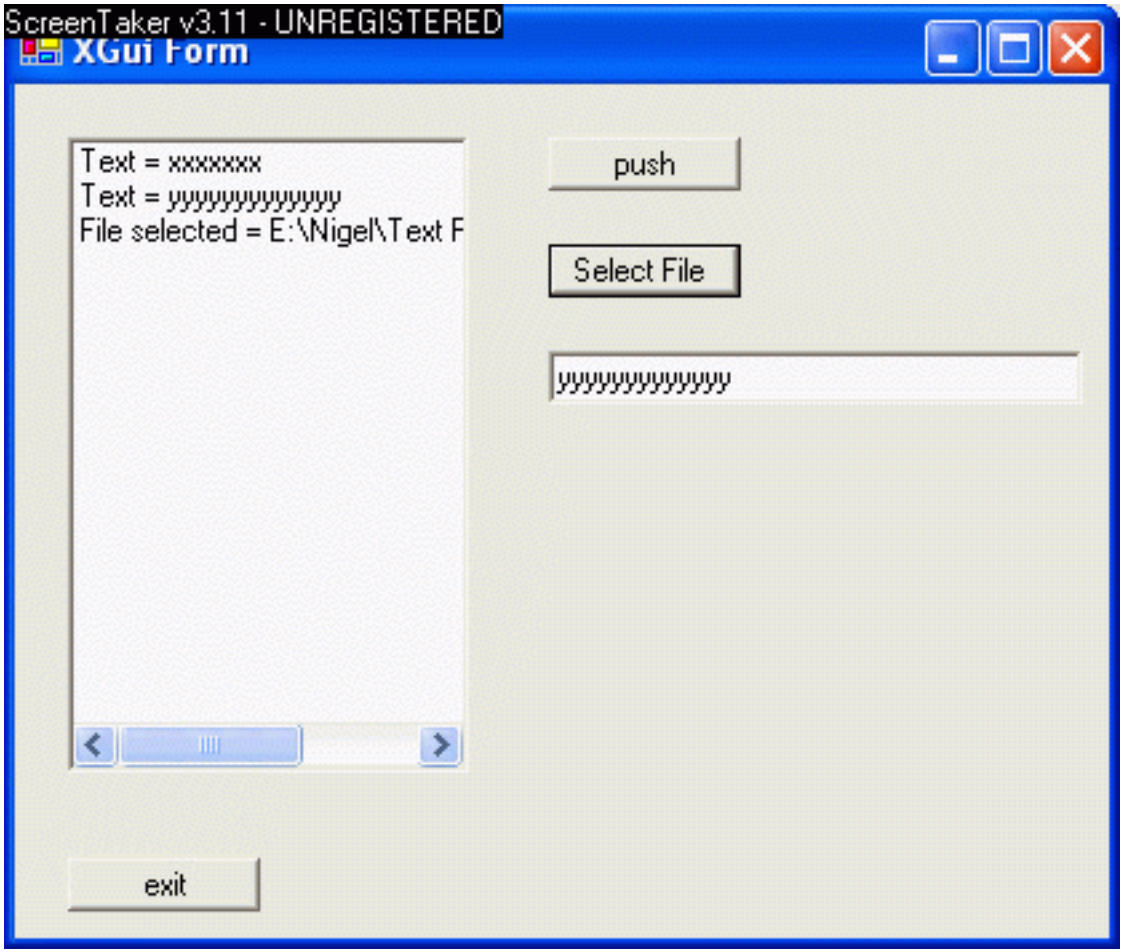
- A Views.VForm class instance is a Windows form on XP and a close simulation of a Windows form on Rotor, but (currently) implemented with Tcl/TK.
- A specification string, used by the class constructor, defines
  - which form controls are wanted, and
  - their layout.
- The specification string is written in XML notation.

Example: a Views.VForm instance

```
string xmlspec = @"<form>
     <horizontal>
       <vertical>
          <listbox name=output/>
          <button name=exit/>
       </vertical>
       <vertical>
         <button name=push width=72/>
         <filedialog name='Select File' width=72/>
         <textbox name=info width=200/>
       </vertical>
     </horizontal> </form>";
Views.VForm gui;

try {
   gui = new Views.VForm(xmlspec);
} catch( Exception e ) {
   // an optional try-catch block: it catches errors in the XML
   Console.WriteLine(e.Message);
   return;
}
```

... and the result looks like this ...



Notes on the XML:

- The implementation uses classes in the System.Xml namespace to parse the XML.
- The XML notation is relaxed in three ways ...
  - capitalization of tagnames and attribute names is ignored,
  - either single or double quotes may be used around attribute values; they may be omitted if the values do not contain special characters, and
  - redundant spaces are eliminated.

  (Classes in the System.Text.RegularExpressions name-space are used to standardize the XML before it is parsed)
- The layout is deliberately kept simple – just vertical lists and horizontal lists (more later).
- There is one tag for each supported forms control.

How do we use the Views.VForm instance?

Interaction is kept very simple so that the form is easily usable by a programmer ...

- no editing of generated code,
- preferably no call-back routines, and
- the minimum number of methods to be used.

Our basic approach is to have the user interact with the form via a simple wait loop ...

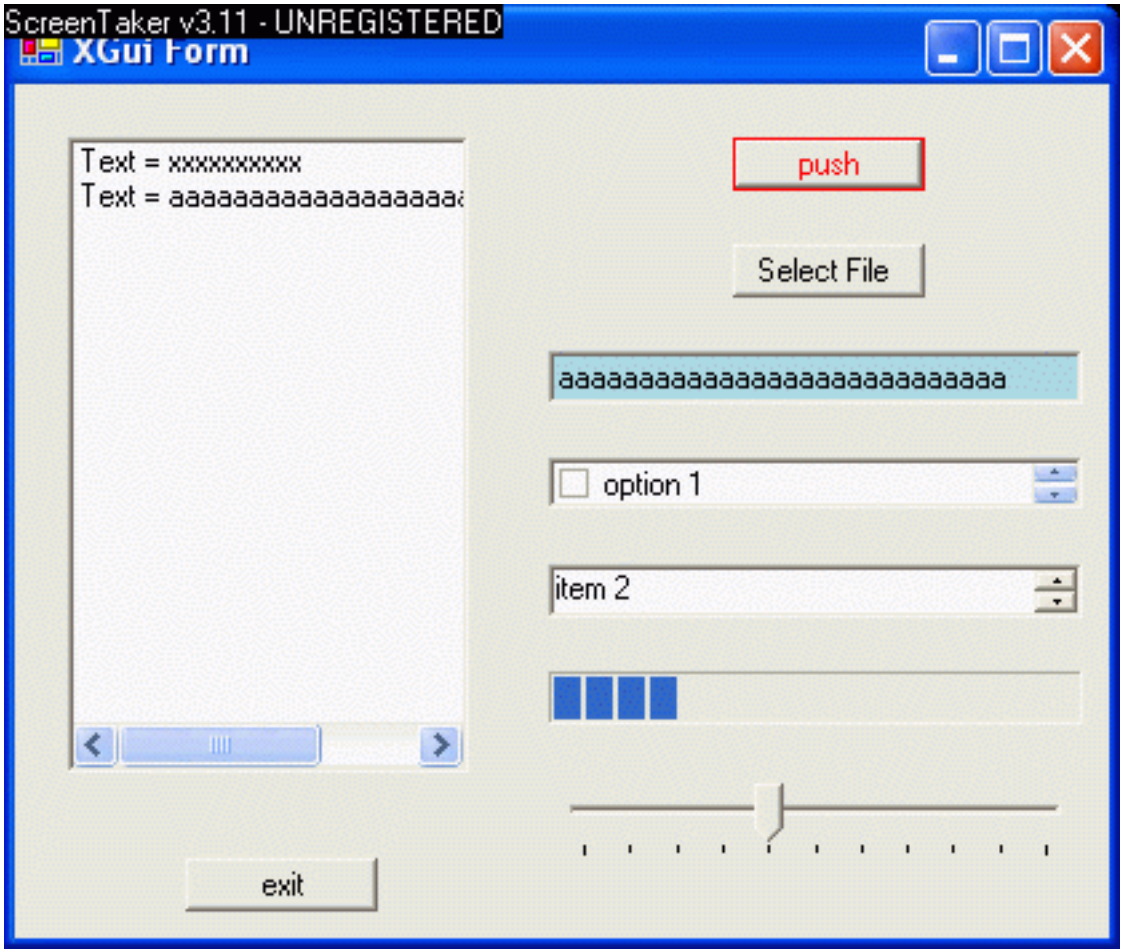
Sample interaction code:

```
// the standard idiom to be imitated by naive programmers
while(gui != null) {

  string b = gui.GetButton();  // wait for a button press

  switch(b) {  // switch on name of button
     case "push":
       string s = gui.GetText("info");
       gui.PutText("output", "Text = " + s);
       break;
     case "exit":
       gui.CloseGUI();
       gui = null; // causes the outer loop to exit
       break;
     case "Select File":
       string f = gui.GetText(b);
       gui.PutText("output", "File selected = " + f);
       break;
  }
}
```

Some more VForm controls & features:



The supported XML constructs (so far) in the Rotor implementation

<form> ... </form>	
<vertical> ... </vertical>	*vertical list of controls / lists*
<horizontal> ... </vertical>	*horizontal list*
<button name=*xxx*/>	
<textbox name=*xxx*/>	
<listbox name=*xxx*/>	
<label name=*xxx*/>	

Supported Attributes (so far)

Name = *xxxx*	*used as a handle*
Text = *xxxx*	
Width = *wwww*	
ForeColor = *cccc*	
BackColor = *cccc*	
HAlign = {left,right,center}	*alignment in vertical list*
VAlign = {top,bottom,middle}	*alignment in horizontal list*

Views.VForm methods (so far) ...

```
Views.VForm( string xmlspec )
```
constructor

```
void CloseGui( )
string GetButton( )
string GetText( string name )
int GetValue( string name )
void PutText( string name, string cval )
void PutValue( string name, int v )
void SetBackColor( string name, Color col )
void SetBackColor( string name, string colorName )
void SetForeColor( string name, Color col )
void SetForeColor( string name, string colorName )
```

Planned or possible additions

- Implementation of more controls.
- Access to more attributes of the controls.
- Explicit placement of controls at X,Y coordinates on a canvas (see next slide).
- Optional event handling functions as callback routines (see next but one slide).

A Form Designer?

- An equivalent of the MDE forms designer tool could generate the XML specification as its result.

  That is why the XML notation needs to include X,Y coordinate specifications, e.g.

```
string xmlspec =
  @"<form>
     <canvas>
       <pos x=10 y=10> <listbox name=output/> </pos>
       <pos x=10 y=30> <button name=exit/> </pos>
       <pos x=100 y=10> <button name=push width=72/> </pos>
       <pos x=100 y=30>
          <filedialog name='Select File' width=72/> </pos>
       <pos x=10 y=50> <textbox name=info width=200/> </pos>
     </canvas>
   </form>";
```

Call-back routines ?

Consider this possible interface ...

```
 string xmlspec = @"<form> <vertical>
                      <button name='Push Me'/>
                      <textbox name='Input'/>
                    </vertical> </form>";
 Views.VForm xg = new Views.VForm(xmlspec);
 xg.RegisterCallBack("Push Me", "pressed", PushMePressed);
 xg.RegisterCallBack("Input", "keystroke", InputChar);

 xg.Run();   // execute until window is closed
 return;
...
 void PushMePressed( string name, Control c ) {
    c.ForeColor = Color.Red; // change button colour
    ...   // do other things
 }

 void InputChar( string name, Control c ) {
    ...   // do various things
 }
```

And other possibilities ...

A control may have a rich variety of fields and methods. It would be nice to provide access to a sophisticated user:

```
Views.Button c = xg.GetControl("Push Me");
c.ForeColor = Color.Blue;
c.BackColor = Color.Red;
c.Width = 100;
c.Text = "Don't Push Me";  // change label on button
```

Note: this is *not* the Windows.Forms.Button class (it is not available with Rotor) but a simplified version of the class implemented in Views.

This work is at an early stage.

Your comments and suggestions are welcome.

C# Simply

Bishop and Horspool

2003