ISVis Adaptation Project

Spring 2009 Term Paper

http://www.cc.gatech.edu/morale/tools/isvis/2009/

Andrew Bernard
College of Computing, Georgia Tech
agbernard@gmail.com

April 30, 2009

Abstract

The objective of this study is to provide an update on the status of the ISVis adaptation. Research done on colormap usage in X Windows is described to elicit a better understanding of the current issues being faced. Along with that we discuss the first refactoring made to the display code and the future plans for continuing the refactoring process.

1 Introduction

This paper describes the work that was continued from the Fall 2008 semester in which we were able to successfully build and run ISVis on Solaris. While the GUI for ISVis could now be displayed, the most outstanding issue was the mural display for the scenario view. The mural for the six components of the main view is working as expected, but not so for the scenario view. In an effort to understand this problem and the overall use of color by the application, some research was done on colormaps and pixmaps in the X Window System - this is discussed in Section 2.

While the colormap issue was being investigated, I also started looking at two other problem areas with the application. Firstly, the primary motivation for my work on this project is to refactor the display code to allow for greater maintainability and flexibility, but up to this point that had not been started. Section 3 talks about the first refactoring we made and the rationale for how we chose it. The other issue was the fact that none of the analysis portion of the application was functioning. Section 4 describes the research we did to try and find alternatives to the original approach employed by ISVis. We will end with a look at some research done on GUI testing and a summary of the project's current status.

2 Debugging the Mural

In order to understand how the mural was being setup and drawn it was important to get a firm grasp on how colors and pixmaps are used in X Windows. The following is a synopsis of chapters 7 and 8 from [7] which discuss colormaps and pixmaps respectively.

2.1 Colormaps

In order to manipulate a color on a display device, we need the following information:

- Pixel value - to designate the color of the pixels on the display
- RGB value - the red/green/blue value of the color
- Visual class - defines the color capability of the display
- Colormap - used to translate between pixel values and colors

There are 3 major visual modes available, each with 2 classes defining whether colormaps are read/write or read-only.

- GrayScale/StaticGray - A pixel value indexes a single colormap that contains monochrome intensities. Each pixel value is represented by up to 8 bits.
- PseudoColor/StaticColor - A pixel value indexes a single colormap that contains color intensities. Each pixel value is represented by up to 8 bits.
- DirectColor/TrueColor - A pixel value is decomposed into separate red, green, and blue subfields. Each subfield indexes a separate colormap and is represented by 8 bits per field, giving each pixel 24 bits total to describe a color.

The visual class being used determines the exact structure of the colormap for any given window in a particular application (each window having a single associated colormap). A colormap is just an array of cells containing a combination of red/green/blue values defining a color. The `XColor` structure is used by X Windows to define the contents of each color cell.

There are three strategies described in this chapter on how to interact with colormaps:

- Shared color cells - like its name suggests, this is simply the strategy of sharing the same color cells across multiple applications.
- Standard color maps - an application can query X Windows for a standard colormap which contains preloaded red/green/blue values for all their color cells.

- Private color cells - allows for the dynamic update of any primary color value stored in the the colormap

ISVis uses the third strategy, creating its own private color map. For this reason it must always be used with a read/write visual class - either PseudoColor (what it was originally coded in) or DirectColor. Consequently it can only be run on displays with those classes available (you can use the 'xdpyinfo' command from the command line to query the display device to see the visual modes that are available). There is an X Windows function, `XDefaultVisual` which can be used to query the display from the code and retrieve a structure, `Visual`, containing the display device's default visual class. We need to make use of this function to check if ISVis can run on the display device it is using. If it can't, we would be able to exit gracefully explaining why ISVis could not execute.

2.2 Pixmaps

A pixmap is a raster - a rectangular area of pixels - like a window, but it is never actually visible. It is used like a buffer to draw the desired shapes into - it can then be copied into a window (which is visible). There are other key differences between a pixmap and a window:

- Background - use `XFillRectangle` instead of `XClearArea` to clear a pixmap
- Borders - pixmaps have no borders because they are never visible
- Colormap - pixmaps have no visual type and consequently no associated colormap. Pixel values in pixmaps do not have colors assigned to them until they are copied to a window

2.3 Mural Status

After studying the above sections I saw no reason why colormaps should not work with a modern (DirectColor) display. So I started slowly adding back in the code I saw that was commented out related to colormaps. The mural is still not displaying properly but I am anticipating this bug will go away as I continue with the process of putting colormaps back in.

3 Refactoring

While I was debugging the mural code, I also started on our first refactoring of the display code. I looked for a portion of the code that I knew executed properly and that could

be easily tested. Another criteria was that it would be as close to "stand-alone" as possible - meaning the candidate would have minimal (if any) references to global variables or methods external to its class. The code pertaining to the color buttons on the main view fit these conditions perfectly. Additionally this code originally caught my attention for a refactoring upon recognizing the Duplicated Code bad smell [6]. I applied the Extract Method refactoring in the following steps:

Step 1: I identified the block of code that was being repeated for each color button. The text highlighted red is the only code that varies per block. This code was originally in `MainView()` lines 241-322.

```
XrmValue bg = X_Application::ColorRmValue("yellow");
XrmValue bg = X_Application::ColorRmValue("#FF70FF");
XrmValue bg = X_Application::ColorRmValue("#7070FF");
XrmValue bg = X_Application::ColorRmValue("cyan");
XrmValue bg = X_Application::ColorRmValue("green");
XrmValue bg = X_Application::ColorRmValue("white");
defaultButton = XtVaCreateManagedWidget(
                    "default",
                    xmPushButtonWidgetClass,
                    canvas,
                    XmNshadowThickness, 2,
                    XmNbackground,
                    (Pixel) *((Pixel *)bg.addr),
                    XmNforeground,
                    X_Application::ColorValue("white"),
                    NULL);
 XtAddCallback(defaultButton, XmNactivateCallback,
            (XtCallbackProc)(&MainView::setSelectedDefaultCB),
            (XtPointer)this);
```

Step 2: I extracted the common block of code into a method called `CreateColorButton` and parameterized anything that needed to vary per method call (i.e. the red text from Step 1).

```
Widget MainView::CreateColorButton(const char * color,
                                   const char * buttonName,
                                   XtCallbackProc callBackProc,
                                   XtPointer callbackMatch)
{
      XrmValue bg = X_Application::ColorRmValue(color);
      Widget button = XtVaCreateManagedWidget(
                        buttonName,
                        xmPushButtonWidgetClass,
                        canvas,
                        XmNshadowThickness, 2,
                        XmNbackground, (Pixel) *((Pixel *)bg.addr),
                        XmNforeground, X_Application::ColorValue("white"),
                        NULL);

      XtAddCallback(button, XmNactivateCallback, callBackProc, callbackMatch);
      return button;
}
```

Step 3: I replaced the 6 blocks of code from Step 1 with 6 method calls to the new method. This reduced the original code from 81 lines to 30.

```
Widget defaultButton =
      CreateColorButton("white", "default", &MainView::setSelectedDefaultCB, this);
Widget purpleButton =
      CreateColorButton("#FF70FF", "   ", &MainView::setSelectedPurpleCB, this);
Widget blueButton =
      CreateColorButton("#7070FF", "   ", &MainView::setSelectedBlueCB, this);
Widget cyanButton =
      CreateColorButton("cyan", "   ", &MainView::setSelectedCyanCB, this);
Widget greenButton =
      CreateColorButton("green", "   ", &MainView::setSelectedGreenCB, this);
Widget yellowButton =
      CreateColorButton("yellow", "   ", &MainView::setSelectedYellowCB, this);
```

Step 4: I extracted the body of the new `CreateColorButton` method to a method of the same name in a new class called `XMainView`. To do this, I added one additional parameter (shown in blue) to the method: the parent (owner) of the button. Previously the parent was a global variable in the `Main_View` class (the canvas variable).

```
Widget XMainView::CreateColorButton(const char * color,
                                    const char * buttonName,
                                    Widget * parent,
                                    XtCallbackProc callBackProc,
                                    XtPointer callbackMatch)
{
      XrmValue bg = X_Application::ColorRmValue(color);
      Widget button = XtVaCreateManagedWidget(
                           buttonName,
                           xmPushButtonWidgetClass,
                           *parent,
                           XmNshadowThickness, 2,
                           XmNbackground, (Pixel) *((Pixel *)bg.addr),
                           XmNforeground, X_Application::ColorValue("white"),
                           NULL);

      XtAddCallback(button, XmNactivateCallback, callBackProc, callbackMatch);
      return button;
}
```

Step 5: I replaced the body of `MainView::CreateColorButton` with a simple call to `XMainView::CreateColorButton`

```
Widget MainView::CreateColorButton(const char * color,
                                   const char * buttonName,
                                   XtCallbackProc * callBackProc,
                                   XtPointer callbackMatch)
{
      return XMainView::CreateColorButton(
                                  color, buttonName, &canvas, callBackProc, this);
}
```

This refactoring will serve as a model for our future design goals of creating an abstract layer for the display code to live under. We will progressively migrate all of the X Windows specific calls out of the main classes and into "X" classes. Once we reach the point where all the main classes are only calling their corresponding "X" classes for X Window related functions, then we will be able to add a layer of abstraction between those calls. Taking our color button method as an example, once we have our abstract layer we would replace our call to `XMain_View::CreateColorButton` with `Abstract_View::CreateColorButton`. Once all of our main classes call our abstract layer we will be free to vary the implementation beneath that layer as needed.

4 Scripts

Currently the static analysis, instrumentation, and trace generation are inoperable due to the lack of a compatible tool to produce the source code information needed for the static analysis. The original method of obtaining this information was to use the Solaris 2.5 CC or cc compiler with the -xsb flag turned on. This would produce files with extra symbol table information for Sun's source browser. ISVis is programmed to read these files and extract the necessary information to produce a static file looking like:

```
 FILE <full filename>
CLASS <class name> <full filename> <line number>
INHERITS <class name> <parent class name>
MEMBER_FUNC <function name> <class name> <full filename> <line number>
GLOBAL_FUNC <function name> <full filename> <line number>
```

We have looked at a number of alternatives ([2], [3], [4], [1]) to replace the outdated Solaris command, but we have not yet decided on or tested any of them. Whichever tool we decide on, the biggest factor to consider is whether or not it produces output that can easily be fed into ISVis for analyzation.

5 GUI Testing Research

In [8], Memon et al. describe the use of an automated oracle to test the expected behavior of a GUI. The automation occurs both in the derivation of what the expected states of the GUI are and in the comparison of those expected states to the actual state of the GUI as it is being executed. The authors achieve this by first creating a model of the GUI's objects, properties, and actions. They then use this model to determine what the expected states of the GUI will be given any particular set of actions. The paper describes three different levels of testing that can be used to compare the expected states to the actual states of the GUI - determining what to use is left up to the test designer.

After the authors lay out how to achieve the automated oracles, they give some performance results showing that the extra time needed to generate the automated test cases is not substantial. Consequently, they argue that even though there is a reasonable amount of human effort involved in creating the oracle, there is definitely an amortized benefit for the work that is done. I am not totally convinced of that argument since it was based on metrics gathered by testing a custom Notepad editor. It would take substantial effort to produce this oracle for ISVis especially considering that it is directly tied to an automated test case generator produced previously by the authors. All of that would have to be done from scratch if ISVis was to take advantage of this method. That being said, if it were in place, it would certainly be beneficial to future development and refactoring.

Another work built on Memon's is that of Qing Xie in [9] where he gives an overview of a

strategy he developed to study GUI faults, GUI event interactions, and the development of GUI fault detection techniques. The rationale behind this is that when testing a GUI an unexpected screen could be generated from an incorrect state of the GUI - in such a case, the tester would want to detect this fault so the test case can be terminated. Xie proposes a framework to help automate the testing process all built on an automatically generated model of the GUI. He then proposes a series of steps that would be needed for the framework to accomplish its purpose.

One module of the framework that would prove useful for ISVis is the Regression Tester. The author does not go into details about how each of the modules are implemented, but just the idea of having a separate entity whose sole purpose is to regression test is a sound one. ISVis could definitely benefit from such a module to verify the refactoring changes are done correctly.

6 Discussion

From my research on GUI testing referenced above and in [5], it seems that the art of testing a GUI is still somewhat specialized based on the needs of a particular application. I have read about efforts made to generalize the art, but I would argue that they would be too elaborate to implement at the present stage of ISVis's life cycle. Honestly, I'm not quite sure what type of testing would best be suited to ISVis so for the moment we are just logging tests in a test plan as seen in Appendix B. We will continue to add to this log until a better (preferably automated) solution presents itself.

As I work with the code more and more, I have been thinking about what exactly would be required for us to consider ISVis "stable". Of course fixing all the bugs that keep it from performing its function would be the first step, but on a broader scale, what does it mean for any application to be considered stable? Does it mean a certain percentage of test cases pass? Do we have to go a certain amount of time without find a bug? Does the code have to be structured a certain way and exhibit certain features? Perhaps it is a function of the percentage of code that is in flux at any given time. This came to my mind when I was deciding on whether or not to start refactoring while I was still finding bugs.

I realize that I have been performing a balancing act between working on the mural bug and proceeding with the refactoring of the color buttons - generally we would find and fix all the bugs first so we don't risk inserting new ones while still trying to fix an old one. That is why I used the criteria that I did to pick the candidate for refactoring because I wanted to be sure to minimize the impact the refactor would have. While it is not completely unrelated code (because the color buttons can be used to set the colors of modules in both the main screen and scenario view) it seemed to not be the cause of the mural code failure.

6.1 Current Status

To summarize the projects current status:

- Display - the application starts and displays the GUI successfully. The mural on the scenario view is not displayed correctly. This is the only observable display related bug.
- Platforms - ISVis runs on any device with DirectColor - currently the application is hard coded to only check for that visual class.

For next semester I plan to finish debugging the mural (and any other display bugs I may find after that). I will also set up the condition to check for the two available visual classes that are needed for ISVis to successfully execute. Hopefully that will go quickly so I can focus most of my efforts on moving all references to X Windows into "X" classes (like `XMainView`) to prepare for the abstract layer.

A Appendix - Code Listing

This is the code from the `Main_View` class that created the color buttons before being refactored.

```
241  // color buttons
242 XrmValue bg = X_Application::ColorRmValue("white");
243 defaultButton = XtVaCreateManagedWidget("default", xmPushButtonWidgetClass,
244                       canvas,
245                       XmNshadowThickness, 2,
246                       XmNbackground,
247                       (Pixel) *((Pixel *)bg.addr),//X_Application::ColorValue("orange"),
248                       XmNforeground,
249                       X_Application::ColorValue("white"),
250                       NULL);
251 XtAddCallback(defaultButton, XmNactivateCallback,
252           (XtCallbackProc)(&MainView::setSelectedDefaultCB),
253           (XtPointer)this);


254
255 bg = X_Application::ColorRmValue("#FF70FF");
256 purpleButton = XtVaCreateManagedWidget("   ", xmPushButtonWidgetClass,
257                     canvas,
258                     XmNshadowThickness, 2,
259                     XmNbackground,
260                     (Pixel) *((Pixel *)bg.addr),//X_Application::ColorValue("#FF70FF"),
261                     XmNforeground,
262                     X_Application::ColorValue("white"),
263                     NULL);
264  XtAddCallback(purpleButton, XmNactivateCallback,
265        (XtCallbackProc)(&MainView::setSelectedPurpleCB),
266        (XtPointer)this);


267
268  bg = X_Application::ColorRmValue("#7070FF");
269  blueButton = XtVaCreateManagedWidget("   ", xmPushButtonWidgetClass,
270                       canvas,
271                       XmNshadowThickness, 2,
272                       XmNbackground,
273                       (Pixel) *((Pixel *)bg.addr),
274//                     X_Application::ColorValue("#7070FF"),
275                       XmNforeground,
276                       X_Application::ColorValue("white"),
277                       NULL);
278  XtAddCallback(blueButton, XmNactivateCallback,
279        (XtCallbackProc)(&MainView::setSelectedBlueCB),
280        (XtPointer)this);
```

```
281
282   bg = X_Application::ColorRmValue("cyan");
283   cyanButton = XtVaCreateManagedWidget("   ", xmPushButtonWidgetClass,
284                         canvas,
285                         XmNshadowThickness, 2,
286                         XmNbackground,
287                         (Pixel) *((Pixel *)bg.addr),
288//                       X_Application::ColorValue("cyan"),
289                         XmNforeground,
290                         X_Application::ColorValue("white"),
291                         NULL);
292   XtAddCallback(cyanButton, XmNactivateCallback,
293         (XtCallbackProc)(&MainView::setSelectedCyanCB),
294         (XtPointer)this);
295
```

```
296   bg = X_Application::ColorRmValue("green");
297   greenButton = XtVaCreateManagedWidget("   ", xmPushButtonWidgetClass,
298                       canvas,
299                       XmNshadowThickness, 2,
300                       XmNbackground,
301                       (Pixel) *((Pixel *)bg.addr),
302//                     X_Application::ColorValue("green"),
303                       XmNforeground,
304                       X_Application::ColorValue("white"),
305                       NULL);
306   XtAddCallback(greenButton, XmNactivateCallback,
307         (XtCallbackProc)(&MainView::setSelectedGreenCB),
308         (XtPointer)this);
309
```

```
310   bg = X_Application::ColorRmValue("yellow");
311   yellowButton = XtVaCreateManagedWidget("   ", xmPushButtonWidgetClass,
312                        canvas,
313                        XmNshadowThickness, 2,
314                        XmNbackground,
315                        (Pixel) *((Pixel *)bg.addr),
316//                      X_Application::ColorValue("yellow"),
317                        XmNforeground,
318                        X_Application::ColorValue("white"),
319                        NULL);
320   XtAddCallback(yellowButton, XmNactivateCallback,
321         (XtCallbackProc)(&MainView::setSelectedYellowCB),
322         (XtPointer)this);
```

B Appendix - Test Plan

Test Number	Test Name	Input	Expected Output	Status
Application				
1	Startup	./iswis <test file>	Application starts and shows the GUI	
Menu Options				
1	Open	File > Open. Then type in session	The six component views populate with data from the saved session.	
2	Open Scenario	Click on trace.info file. Click Scenario > View	A new window appears showing the scenario view for the selected trace file. A visualization (known as a mural) should show at the right of the new window depicting the entire scenario from top to bottom.	Mural not displaying.
Color Buttons				
1	Yellow Button	Click to highlight any item in any of the six main component views. Click the yellow button.	The highlighted item is highlighted yellow.	
2	Green Button	Click to highlight any item in any of the six main component views. Click the green button.	The highlighted item is highlighted green.	
3	Cyan Button	Click to highlight any item in any of the six main component views. Click the cyan button.	The highlighted item is highlighted cyan.	
4	Blue Button	Click to highlight any item in any of the six main component views. Click the blue button.	The highlighted item is highlighted blue.	
5	Purple Button	Click to highlight any item in any of the six main component views. Click the purple button.	The highlighted item is highlighted yellow.	
6	White Button	Click to highlight any item in any of the six main component views. Click the white button.	The highlighted item is highlighted white.	

C Time Log

Week	Hours	Activity
1/5 - 1/11	0	No Meeting.
	4.5	Updating the makefile to get it to work on different OS's, specified by the target to the make command.
	1	Debugging the code to get the display to show on solaris.
Total:	**5.5**	
1/12 - 1/18	1	Meeting
	9	Debugging the code to get the display to show on solaris.
Total:	**10**	
1/19 - 1/25	0	No Meeting - MLK Day.
	1	Changed the makefile so there is a separate file for each OS type, called by the main makefile depending on what target it is given.
	10	Debugging the code to get the display to show on solaris.
Total:	**11**	
1/26 - 2/1	1	Meeting
	2	Researching X Windows.
	2	Working on a Hello World X Windows application.
	1	Updated the code to properly trap and report signals.
	3	Debugging the code to get the display to show on solaris
	3	Reading and summarizing [13]
	3	Reading and summarizing [14]
Total:	**15**	
2/2 - 2/8	0	No Meeting
	1	Finished my Hello World X application.
	1	Researching X Windows.
	8	Debugging the code to get the display to show on solaris.
	3	Reading and summarizing [15]
Total:	**13**	

2/9 - 2/15	1	Meeting
	1.5	Researching X Windows.
	12	Debugging the code to get the display to show on solaris.
	3	Reading and summarizing [16]
Total:	**17.5**	
2/16 - 2/22	1	Meeting
	5	Debugging: the display is showing up (finally!) on solaris, but some of the menu items are causing seg faults.
	3	Got ISVis to run on Linux (Ubuntu on my laptop)
Total:	**9**	
2/23 - 3/1	1	Meeting
	1	Debugging: trying to figure out why the display is not showing the mural.
Total:	**2**	
3/2 - 3/8	0	No Meeting.
	2	Prepared updated version of Fall 2008 paper.
Total:	**2**	
3/9 - 3/15	1	Meeting
Total:	**1**	
3/16 - 3/22	0	No meeting - spring break.
	2.5	Reading and summarizing [17]
	3	Learning about color maps and visual modes in X windows
	1	Summarizing [19]
	10	Debugging the mural code.
	1	Researching alternatives to the script used to generate the trace file.
Total:	**17.5**	
3/23 - 3/29	1	Meeting
	1	Debugging the mural code.
	1.5	Reading and summarizing [18]
Total:	**3.5**	

3/30 - 4/5	0	No Meeting
	8	Debugging the mural code.
	1	Reading and summarizing [20]
Total:	**9**	
4/6 - 4/12	1	Meeting
	8	Debugging the mural code.
	2	Refactoring the code for color buttons.
Total:	**11**	
4/13 - 4/19	1	Meeting
	3	Refactoring the code for color buttons - this time creating the XMainView class to hold all the references to X made by MainView.
Total:	**4**	
4/20 - 4/26	0	No Meeting
	3	Working on term paper.
Total:	**3**	
4/27 - 5/1	0	No Meeting
	9	Working on term paper.
Total:	**9**	

Total Time: 143

References

[1] Lattix. http://www.lattix.com.

[2] Project bauhaus. http://www.bauhaus-stuttgart.de/bauhaus/index-english.html.

[3] Semantic designs. http://www.semanticdesigns.com/Products/Metrics/index.html.

[4] Understand. http://www.scitools.com/products/understand/features.php.

[5] Andrew Bernard. Isvis adaptation project - fall 2008. December 2008.

[6] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[7] Oliver Jones. *Introduction to the X Window System*. Prentice Hall, 1989.

[8] Atif M. Memon, Martha E. Pollack, and Mary Lou Soffa. Automated test oracles for guis. *SIGSOFT Softw. Eng. Notes*, 25(6):30–39, 2000.

[9] Qing Xie. Developing cost-effective model-based techniques for gui testing. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 997–1000, New York, NY, USA, 2006. ACM.