A Cabot Technical Paper:

DTV UI Differentiation Using SVG

Typical UI Challenges in the Digital Television Market

In the increasingly crowded and competitive digital receiver marketplace, the ability to create strongly branded, differentiated products is critical for manufacturers to attain market advantage. Although programming languages like C and C++ allow the implementation of almost any graphical treatment; limited engineering resource and time to market pressures have meant that graphical changes attempted by many manufacturers are basic i.e. colours, buttons and logos.

Presently User Interface (UI) designers must deliver mock-up UIs and the developer must realise these interfaces in the code. The designer is not free to experiment and the developer must reproduce the layout systematically and laboriously.

Third party UI tools are more often proprietary and require extensive training and support. They incur either a third party royalty and/or high tool cost, which would prevent the manufacturer from providing the most cost effective solution to its customers. Today's high-end products will inevitably become tomorrow’s low-end products and more price sensitive.

Developing an entire UI application in a new language would mean disposing of existing code and committing more development resource, a commodity that can be ill-afforded in today’s marketplace.

Simple SVG Solutions to UI Challenges

Scalable Vector Graphics (SVG) provides a solution to many of these problems. SVG is a family of specifications of Extensible Markup Language (XML) based file formats for describing two-dimensional vector graphics, both static and dynamic (interactive or animated). The SVG specification is an open standard that has been under development by the World Wide Web Consortium (W3C) since 1999.

As an open standard there are many resources and tools available; and as an XML based technology it is easy to manipulate in almost any language. In addition, a SVG based solution can preserve existing robustly tested C++ functionality contained in legacy applications. SVG can also be made simple enough for designers to use, yet flexible enough to enable them to explore creative possibilities.

Traditionally, many graphical resources and dialog layouts can become locked into specific implementations. User interfaces described in SVG offer the opportunity to be used more readily in future technology, whether this involves being manipulated to another format using a tool or allowing software to be easily developed to read the resources directly themselves.

SVG in action at Cabot Communications

A simple SVG renderer was built on top of the Cabot's existing rendering engine within the UI framework. Use of the existing engine constrained what could be implemented in SVG, but provided a working subset of the features required to make SVG useful and reduce development time. Support was provided for lines, polygons, circles, ellipses, paths, and linked PNG files; using a 256 colour model with standard and High Definition screen resolutions.

One of the main objectives was to abstract the engine away from the proprietary concepts of the existing UI framework. Interfaces were put in place to provide mapping to the palette, image, and font resources that would be used to support SVG rendering. A drawing façade interface allowed the underlying Cabot rendering engine to be driven without coupling the implementation.

In order to facilitate the widest possible use of SVG tools, the decision was made not to extend the XML syntax used. The principle was to "keep it simple" as much as possible. To this end, only 'id' attributes were used to integrate with Cabot applications. The 'id' attribute provides a simple mechanism for identifying the position and size of SVG elements, and can be used to bind the SVG to the C++ concepts at runtime. All SVG tools support the manipulation of the id attribute, and it is an easy concept to communicate to users. Naming conventions allow these SVG elements and their properties to be used in the C++ software. The simple Cabot renderer fills a collection of "named rectangle" shapes, those shapes with an id attribute, whilst performing its rendering task. These "named rectangle" shapes are not rendered, but available for the C++ code to interrogate their properties.

Rather than embed all the SVG required to render a dialog, sets of SVG files were developed to associate with a group of widgets. SVG files are referred to in the C++ code to bind the SVG elements at runtime. This provides a clear and relatively flat interface for designers and developers to work with. This was a very important objective as the Cabot Digital Television Recorder (DTR) application itself consists of over 70 dialogs and menus, represented by 150 separate SVG files.

The next task was to develop a widget set. Cabot's requirement focused on the need to make it easy to port existing UI framework applications. The UI framework gave us a ready made widget set to act as a basis for implementation. The widget set includes:

- TextBox
- Button
- SelectionBox
- ListBox
- Progressbar
- Scrollbar
- Menu
- TextEntryBox

The UI framework widgets have a simple focused or unfocused state. This model was adequate when the UI framework was first developed but since that time applications have grown increasingly complex. The SVG implementation extended this focused/unfocused model, to give the designer more flexibility with the graphical styles. As a result, each widget in the SVG implementation can have one of four widget states. The transitions between these states are shown in

the state model below. This state model is driven by the C++ framework in which the widget set is implemented.

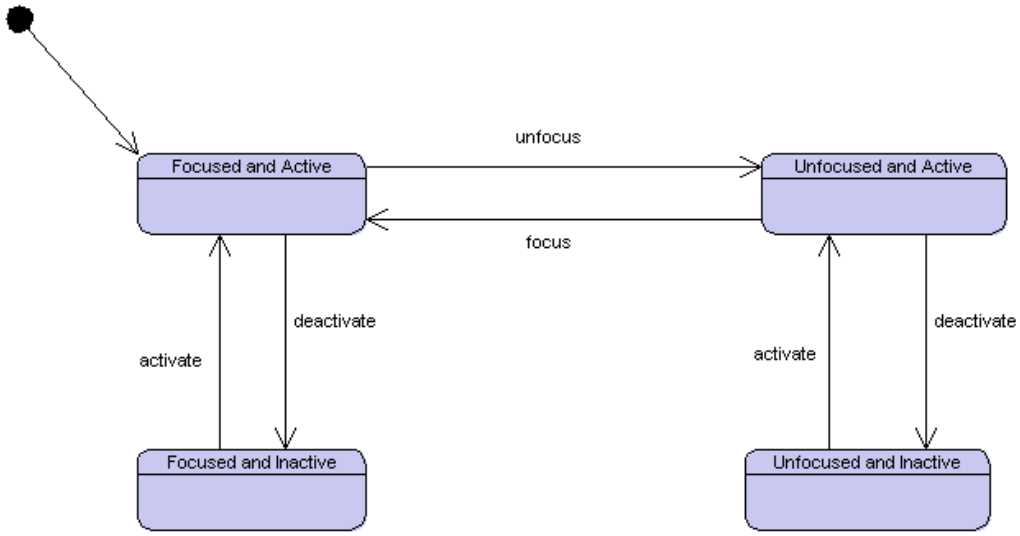

Figure 1 - Widget State Model

When a widget is rendered as part of a dialog, a discovery mechanism finds the SVG files associated with each state. Within each of these SVG files there are groups of graphical elements that allow the software to resize each widget to the required size. The widget requests the rendering of each group at the required position, providing an appropriate scaling transform for each group. For example, a button widget has a default SVG file "buttonFA.svg" that provides the look of the widget when it is in a focused active state. The SVG contains three groups, each group has a name and describes a part of the button - the left end, middle, and right end.

Figure 2 - Button Groups

When Simple button widgets are rendered the groups "gRowL" and "gRowR" are rendered at the end of the available space without any scaling. The "gRowM" group is scaled horizontally to fill the space in between "gRowL" and "gRowR".

The Cabot renderer allows graphical elements to be scaled and rendered relative to named rectangles described earlier. Using this concept rather than masks

makes the implementation of the SVG renderer much simpler, and allows the position of such masks to be identified more clearly in graphics tools.

A button SVG file has an indication of the position and attributes of the text that renders the button label. We use a rectangle "textPos" which is part of the "gRowM" group and describes the attributes of the text to be rendered. Other attributes include: stroke colour, fill colour and height (font size).

Each of our widgets use similar techniques to allow groups and masks to scale graphics to the appropriate size, even horizontally and vertically. When a dialog is displayed, the associated SVG file is rendered, providing the background graphics using "named rectangles" describing the position and attributes of each widget within the dialog. The figure below shows how an example of an SVG templated dialog.

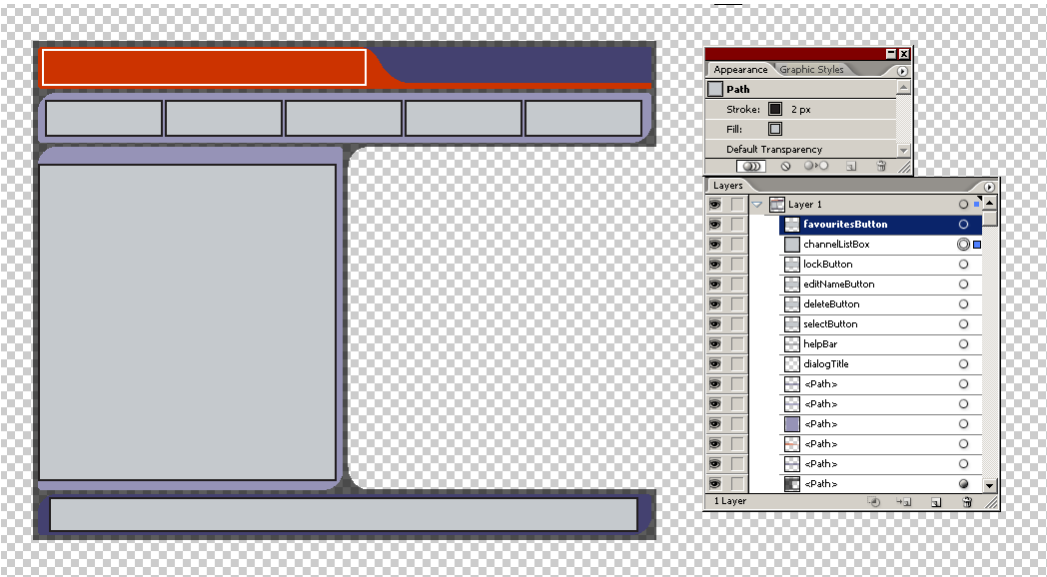

Figure 3 - Channel List dialog template

Each rectangle that indicates widget position is given a name. The Software Developer uses these names in the C++ for the dialog to bind to the required widget at runtime. e.g. "dialogTitle" represents the position and size of the dialog title. The stroke/fill is used to determine colours and text size. The C++ itself defines the text to be displayed. Here the widget provides the internationalised text using the efficient mechanisms provided by the Cabot UI framework.

This method of dialog implementation couples the dialog SVG file to the C++ implementation, the C++ side relying on certain elements to be present in the SVG. Whilst this restricts the UI designer, it does ensure that the graphical treatment is complete, making verification by the customer easier.

These techniques provide a very simple method for both converting existing applications and developing new applications with SVG. To convert an existing application the developer simply has to replace the existing widgets and dialogs with their SVG equivalents, and provide simple SVG templates. Designers can then apply a graphical treatment to the SVG templates separately from the software development.

The End Result

Rendering a radically different graphical treatment for an existing DTR application can easily consume three months' worth of engineering effort. Using the new SVG method for producing graphical treatments for a DTR application requires just two to three weeks of a designer's effort and minimal engineering support, a dramatic saving in engineering effort and time to market. The perceived quality of the UI is greatly enhanced when graphical design experts are used. Greater re-use of graphical resources is also encouraged through the techniques offered by commercial graphics tools and robustly tested legacy applications can be deployed to different customers with radically different graphical treatments.

Future Development

The success of the first simple deployment of SVG technology has paved the way for Cabot to develop a fully compliant HD SVG engine, including animation to further enhance the UI experience. The powerful scripting support in SVG enables dynamic applications to be created. The simple naming conventions used open up the prospect for code generation and round trip engineering concepts to be provided by an integrated toolset. SVG-based UI solutions create a much more open, empowering development environment where manufacturers and end-customers can benefit alike.

If you would like any further information regarding the topic covered in this paper, please contact info@cabot.co.uk

CABOT

Cabot Communications Ltd, Verona House, Filwood Road, Bristol, BS16 3RY, UK
Tel: +44 (0) 117 958 4232 Fax: +44 (0) 117 958 4168 Email: info@cabot.co.uk