International Journal of Computer and Electrical Engineering, Vol. 1, No. 4, October, 2009
1793-8163

Developing device independent visual components for web applications using Affine Vector Graphics and Silverlight framework

Appasami.G and Suresh Joseph. K

***Abstract* - Most of the languages support Visual components for a particular platform only. These components and Graphical user interface (GUI) controls are dependent on specific operating system, browser and resolution. Silverlight is one of the Client side technologies in Dot Net to develop rich internet applications. Silverlight provides independent of operating system and browser specific Components and controls. In this paper we are mainly concentrating on resolution independent Visual components. These components will perfectly fit to any browser with m x n resolutions in LCD, CRT, PDA and other devices within any operating systems including Linux. These components and controls are placed with relative position not by absolute position. These controls position, length, height, appearance, effects, style, etc are adjusted in viewing device. The new components and controls are developed from Silverlight base classes by inheriting all properties and methods. Silverlight controls are developed to support all kinds of browser and for all major operating systems. XAML is new Microsoft open standard markup language used in Silverlight to describe GUI. So in this paper we are concentrating on resolution independent Visual components for web application.**

***Index Terms* - Affine vector Graphics, User Interface, Silverlight, resolution, device independent Visual components and Markup languages.**

I. INTRODUCTION

Resolution independent visual components refers that any component that should be compatible to all kinds of monitor with different resolutions [8][9]. A visual component is nothing but a collection graphical user interfaces. The benefit of Resolution independent visual components is that they can scale to any size without losing quality. Suppose a textbox takes 20*100 pixels in 800*600 resolution monitor then it should be automatically fit to any size of monitor. For example in 1600*1200 resolution monitor the same text box size should be adjusted to 40*200 pixels [1][2][3]. Here the both monitors resolution ratio is 1:2. But in general the resolution ratio varies in large. So adjusting the controls itself very difficult. Text can be displayed in any device without any intermediate conversion. But rendering graphical user interface to different display are not easy task. A user interface button can be rendered to any device, but it should be modified and rendered to a particular device.

Since Silverlight supports vector graphics we can surely develop device independent visual components, but in web technology, lots of UI controls are built on non vector graphics.

Resolution Independent User Interface refers to the ability of the UI rendering engine to use vector graphics to draw primitives; lines, curves, shapes, color, etc. The benefit of vector graphics is that they can scale to any size without losing quality.

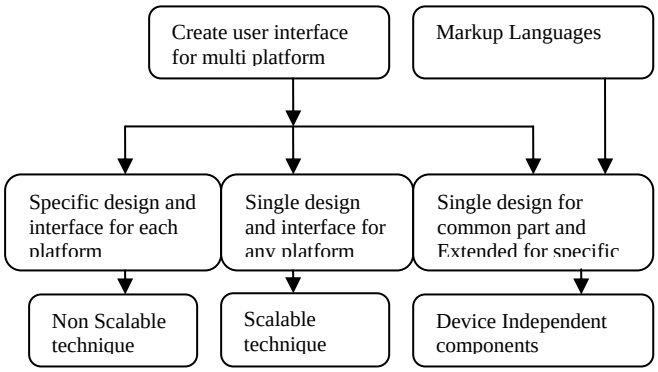

Figure 1. Device Independent Components Architecture.

Computer displays are made up from small dots called pixels. The picture is built up from these dots. The smaller and closer the dots are together the better the quality of the image but the bigger the file needed to store the data. If the image is magnified it becomes grainy as the resolution of the eye enables it to pick out individual pixels. Vector graphics files store the lines, shapes and colors that make up an image as mathematical formulae. A vector graphics program uses the mathematical formulae to construct the screen image by building the best quality image possible, given the screen resolution, from the mathematical data. The mathematical formulae determine where the dots that make up the image should be placed for the best results when displaying the image. Since these formulae can produce an image scaleable to any size and detail the quality of the image is only determined by the resolution of the display and the file size of vector data generating the image stays the same. Device independent visual components can be done in Silverlight by using vector graphics and markup languages [21][22][23].

Device independent Visual components can be developed

Manuscript received June 26, 2008.

Appasami. G is with the Department of computer Science, pondicherry University, Pondicherry, India (phone: +91-9786554175).

Suresh Joseph. K is with the Department of computer Science, pondicherry University, Pondicherry, India (phone: +91-9444321492).

.

in Dot Net Environment using Silverlight markup language XAML and Affine vector graphics. Developing device independent Visual components using GUI is elaborately described in this paper [18][19][20].

II. Affine Vector Graphics

The most common properties of vector graphics are resolution independence, loss less rendering and to any size. The operations of vector graphics like translation, rotation, magnification and minification. These operations are also called as geometric operations. If a graphics supports translation, rotation, magnification and minification then it is called affine vector graphics [4][5][6].

A. Translation, minification, magnification, and rotation

Image translation, scaling, and rotation can be analyzed from a unified standpoint. Let G(j, k) for 1<j<J and 1<k<K denote a discrete output image that is created by geometrical modification of a discrete input image F(p,q) for 1<p<P and 1<q<Q. In this derivation, the input and output images may be different in size. Geometrical image transformations are usually based on a Cartesian coordinate system representation in which the origin (0,0) is the lower left corner of an image, while for a discrete image, typically, the upper left corner unit dimension pixel at indices (1, 1) serves as the address origin. The relationships between the Cartesian coordinate representations and the discrete image arrays of the input and output images are illustrated in Figure 2. The output image array indices are related to their Cartesian coordinates by

$$x_k = k - \tfrac{1}{2}$$
$$y_k = J + \tfrac{1}{2} - j \qquad (2.1)$$

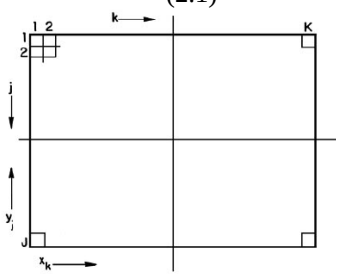

FIGURE 2. Relationship between discrete image array and Cartesian coordinate representation.

Similarly, the input array relationship is given by

$$u_q = q - \tfrac{1}{2}$$
$$v_p = P + \tfrac{1}{2} - p \qquad (2.2)$$

Translation: Translation of F(p,q) with respect to its Cartesian origin to produce G(j,k) involves the computation of the relative offset addresses of the two images. The translation address relationships are

$$x_k = u_q + t_x$$
$$y_j = v_p + t_y \qquad (2.3)$$

Where $\mathbf{t_x}$ and $\mathbf{t_y}$ are are translation offset constants. There are two approaches to this computation for discrete images: forward and reverse address computation. In the forward approach, $\mathbf{u_q}$ and $\mathbf{v_p}$ are computed for each input pixel (p,q) and substituted into Eq. 2.3 to get $\mathbf{x_k}$ and $\mathbf{y_j}$ Next, the output array addresses (j,k) are computed by inverting Eq. 2.1. The composite computation reduces to

$$j' = p - (P - J) - t_y$$
$$k' = q + t_x \qquad (2.4)$$

where the prime superscripts denote that **j'** and k' are not integers unless $\mathbf{t_x}$ and $\mathbf{t_y}$ are integers. If **j'** and **k'** are rounded to their nearest integer values, data voids can occur in the output image. The reverse computation approach involves calculation of the input image addresses for integer output image addresses. The composite address computation becomes

$$p' = j + (P - J) + t_y$$
$$q' = k - t_x \qquad (2.5)$$

where again, the prime superscripts indicate that p' and q' are not necessarily integers. If they are not integers, it becomes necessary to interpolate pixel amplitudes of F(p,q) to generate a re sampled pixel estimate $\tilde{F}(p,q)$, which is transferred to G(j,k).

Scaling: Spatial size scaling of an image can be obtained by modifying the Cartesian coordinates of the input image according to the relations

$$x_k = s_x u_q$$
$$y_j = s_y v_p \qquad (2.6)$$

Where $\mathbf{s_x}$ and $\mathbf{s_y}$ are positive-valued scaling constants, but not necessarily integer valued. If $\mathbf{s_x}$ and $\mathbf{s_y}$ are each greater than unity, the address computation of Eq. 2.6 will lead to magnification. Conversely, if $\mathbf{s_x}$ and $\mathbf{s_y}$ are each less than unity, minification results. The reverse address relations for the input image address are found to be

$$p' = (1/s_y)(j + J - \tfrac{1}{2}) + P + \tfrac{1}{2}$$
$$q' = (1/s_x)(k - \tfrac{1}{2}) + \tfrac{1}{2} \qquad (2.7)$$

As with generalized translation, it is necessary to interpolate F(p,q) to obtain G(j,k).

Rotation: Rotation of an input image about its Cartesian origin can be accomplished by the address computation

$$x_k = u_q \cos\theta - v_p \sin\theta$$
$$y_j = u_q \sin\theta + v_p \cos\theta \qquad (2.8)$$

Where q is the counterclockwise angle of rotation with respect to the horizontal axis of the input image. Again, interpolation is required to obtain G(j,k). Rotation of an input image about an arbitrary pivot point can be accomplished by translating the origin of the image to the pivot point, performing the rotation, and then translating back by the first translation offset. Equation 2.8 must be inverted and substitutions made for the Cartesian coordinates in terms of the array indices in order to obtain the reverse address indices (p',q'). This task is straightforward but results in a messy expression. A more elegant approach is to formulate the address computation as a vector-space manipulation.

B. . Generalized Linear Geometrical Transformations

The vector-space representations for translation, scaling, and rotation are given below.

Translation:

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} u_q \\ v_p \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (2.9)$$

Scaling:

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix} \quad (2.10)$$

Rotation:

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix} \quad (2.11)$$

Now, consider a compound geometrical modification consisting of translation, followed by scaling followed by rotation. The address computations for this compound operation can be expressed as

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix} + \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (2.12)$$

On consolidation,

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} s_x \cos\theta & -s_y \sin\theta \\ s_x \sin\theta & s_y \cos\theta \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix} + \begin{bmatrix} s_x t_x \cos\theta & -s_y t_y \sin\theta \\ s_x t_x \sin\theta & +s_y t_y \cos\theta \end{bmatrix} \quad (2.13)$$

It can be linearly expressed as

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} c_0 & c_1 \\ d_0 & d_1 \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix} + \begin{bmatrix} c_2 \\ d_2 \end{bmatrix} \quad (2.14)$$

This can be rewritten in the more compact form

$$\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} c_0 & c_1 & c_2 \\ d_0 & d_1 & d_2 \end{bmatrix} \begin{bmatrix} u_q \\ v_p \\ 1 \end{bmatrix} \quad (2.15)$$

The three-dimensional vector representation of a two-dimensional vector is a special case of a *homogeneous coordinates* represented as

$$\begin{bmatrix} x_k \\ y_j \\ 1 \end{bmatrix} = \begin{bmatrix} c_0 & c_1 & c_2 \\ d_0 & d_1 & d_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_q \\ v_p \\ 1 \end{bmatrix} \quad (2.16)$$

Equation 2.16 is used to match from one image to another with all geometric operations like rotation, magnification and rotation [4][5][6].

C. . Affine Transformation

The geometrical operations of translation, size scaling, and rotation are special cases of a geometrical operator called an *affine transformation*. It is defined by Eq. 2.14 in which the constants *ci* and *di* are general weighting factors. The affine transformation is not only useful as a generalization of translation, scaling, and rotation. It provides a means of image shearing in which the rows or columns are successively uniformly translated with respect to one another. The rendering device can easily render User interfaces to any device using affine transform. Translation and scaling are very essential to render user interface to different kinds of devices [1][2][4][5].

The figure 3 shows affine transformation like translation, rotation, minification and magnification with textbox and button controls [1][4][5].

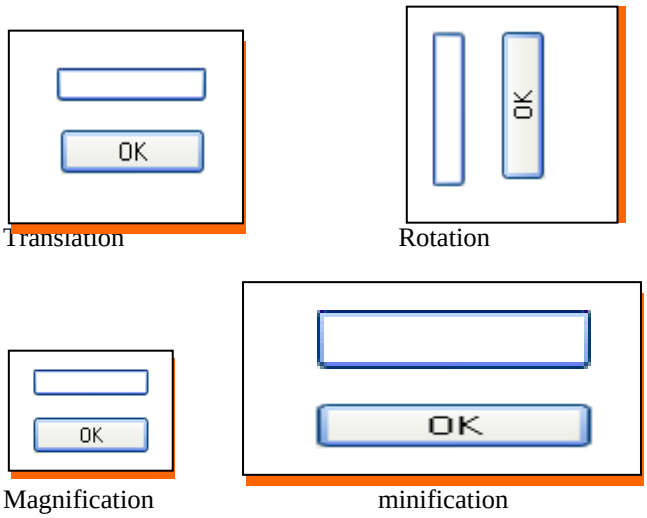

Figure 3. (a). Translation, (b). Rotation, (c). minification (d). Magnification

Suppose these controls are developed for a particular terminal with bitmap image support, these controls look like in figure 4 for translation, rotation, minification and magnification. The quality of Graphical user interface controls will become very worst. Controls developed with bitmaps have lot of disadvantages. So it is not suitable for developing device independent controls with bitmap images [1][2][4][5].

Disadvantages of Bitmaps

- Bitmaps are resolution dependent.
- Consumes more memory while processing.
- File size too large to hold bitmaps.
- Quality will be less while rendering to devices.

Example of translation, rotation, minification and magnification of bitmaps

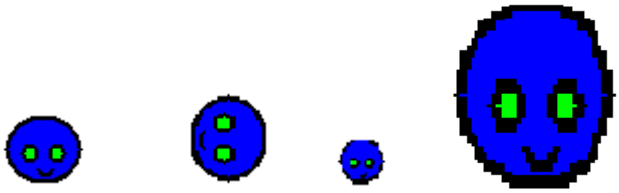

Figure 4. (a). Translation, (b). Rotation, (c). minification (d). Magnification

Example of translation, rotation, minification and magnification of affine vector Graphics.

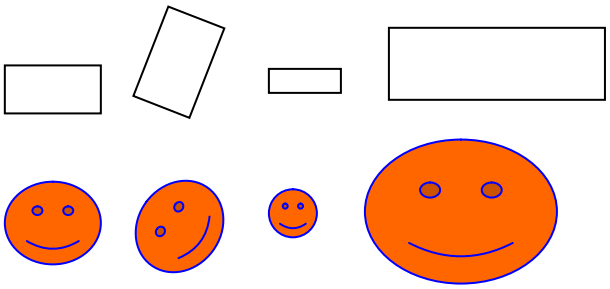

Figure 5. Affine vector Graphics. (a). Translation, (b). Rotation, (c). minification (d). Magnification

Advantages of Vector Graphics

- Scalability
- File size based on complexity rather than color depth
- Easily generate by programs
- Components can be individually manipulated
- Processing done on client
- Potential for animation, interaction
- Device Independent resolution controls

Since vector graphics preserve images, it is best suitable for developing device independent visual components. Affine vector graphics always preserve images with good quality after any transformations like translation, rotation, minification and magnification [2][6][7]. Figure 5 shows translation, rotation, minification and magnification using Affine vector graphics. Now we can say by seeing figure 4 and figure 5 affine vector graphics is best choice for developing device independent visual components [1][2][10][12].

III. MARKUP LANGUAGES

A markup description is higher than toolkit programming in the user interface abstraction. Markup language follows open standards. It is easy to understand and writing. Markup languages were originally invented for describing and preserving data. With the advent of the Web, markup languages are now also being used to describe and preserve user interfaces. They provide high degrees of portability and this allows cross platform user interfaces to be distributed over the Internet. Unlike traditional programming, markup languages require little programming experience and are usable by novice programmers. Markup descriptions can also be generated from visual builders, thus removing the need for memorizing the markup language's vocabulary. Almost all the new markup languages are applications of the extensible Markup Language (XML). The W3C provides the general syntax and each language provides the vocabulary and semantics. Markup languages adhering to the XML format are applicable to a wide range of applications, including databases, web development, searching, and user interfaces [8][17][18].

Markup descriptions can be distributed to new platforms without additional processing. Unlike imperative Programming code, they do not require compilation. They also take less storage space than compiled code, which makes them faster to download over a network. For example, an HTML interface downloads faster than a Java interface. Markup descriptions are stored in pure text and are very resistant to bit-errors. For example, if there is a single-bit error in a compiled (binary) program, then the application might be unusable. However, if there is a single-bit error in a markup file, then the damage to the application is very small and in many cases recoverable [17][18].

Markup descriptions are usually device-independent. The device-dependent information is stored in a separate description called a style sheet. The word "style" refers to the mapping between the markup tags and the semantics. There are several proposed standards for describing style information: XSL, CSS, DSSSL, etc [11][13][17][18].

A. . HyperText Markup Language (HTML)

Web pages are a special category of GUIs. They are the most popular kind of interface for Internet applications. A Web page is simply a markup document (written in HTML) that is downloaded and interpreted on the client by an HTML-capable browser. HTML is now succeeded by XHTML. XHTML conforms to the XML format. Both HTML and XHTML use the page metaphor to present the information to the user [11][13].

B. Extensible Markup Language (XML)

XML is a general-purpose specification for creating custom markup languages. It is classified as an extensible language, because it allows the user to define the mark-up elements. XML's purpose is to aid information systems in sharing structured data, especially via the Internet, to encode documents, and to serialize data, in the last context, it compares with text-based serialization languages such as JSON, YAML and S-Expressions. XML's set of tools helps developers in creating web pages but its usefulness goes well beyond that. XML, in combination with other standards, makes it possible to define the content of a document separately from its formatting, making it easy to reuse that content in other applications or for other presentation environments. Most importantly, XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations without needing to pass through many layers of conversion.

XML began as a simplified subset of the Standard Generalized Markup Language (SGML), meant to be readable by people via semantic constraints; application languages can be implemented in XML. These include XHTML, RSS, MathML, GraphML, Scalable Vector Graphics, MusicXML, and others. Moreover, XML is sometimes used as the specification language for such application languages. XML is recommended by the World

Wide Web Consortium (W3C). It is a fee-free open standard. The recommendation specifies lexical grammar and parsing requirements [11][13].

C. Motif User Interface Language (UIL)

The User Interface Language (UIL) is a grammar for statically describing the layout of widgets. It is easier to describe the layout of widgets than to write the repetitive code to call all the functions to create and layout the widgets. UIL files have the extension .uil. They are compiled to files with the extension .uid (uid stands for User Interface Definition). A C program calls functions in the Mrm (Motif Resource Manager) library, which opens and reads the content of a .uid file. As a .uid file is read, the widgets it describes are created [11][13].

D. Extensible User Interface Language (XUL)

XUL is an XML-based language for describing the contents of windows and dialogs. XUL was created by the Mozilla community to simplify the user interface development for new applications running under the Netscape Web browser. XUL has language constructs for all of the typical dialog controls, as well as for widgets like toolbars, trees, progress bars, and menus [11][13].

E. User Interface Markup Language(UIML)

The User Interface Markup Language (UIML) is an XML-based language whose goal is to express user interfaces for multiple software platforms on different devices and for multiple applications. One thing that sparked the research that lead to the design of UIML2 was a performance analysis tool called Chitra. It was originally written as a big monolithic program, a design that made maintenance and updating very difficult. Following proper software engineering principles, Chitra was then broken into multiple independent pieces, and interaction was done through command-line interfaces. Each piece was designed, implemented, debugged, and maintained in isolation. New pieces could easily be added without any changes to the rest of the application. When Chitra finally got a graphical user interface, it was no longer possible to add new pieces without modifying the code for the user interface [13].

Maintaining the GUI code means the UI programmer must understand both the graphical toolkit and the entire application. In the case of Chitra this was very difficult, since as with any academic software, there were no permanent programmers. New programmers must learn the graphical toolkit and review the entire application design before they can make any additions to the interface. That is when the idea arose of having the user interface automatically generated from a language that is simple enough to be used by novice programmers, yet powerful enough to handle most user interfaces [11][13].

F. Extensible Application Markup Language (XAML)

Extensible Application Markup Language (XAML, pronounced zammel) is a declarative XML-based language created by Microsoft which is used to initialize structured values and objects. It is available under Microsoft's Open Specification Promise. The acronym originally stood for Extensible Avalon Markup Language - Avalon being the code-name for Windows Presentation Foundation (WPF) [11][14][15].

XAML is used extensively in .NET Framework 3.0 technologies, particularly Windows Presentation Foundation, Silverlight, and Windows Workflow Foundation (WF). In WPF, XAML is used as a user interface markup language to define UI elements, data binding, eventing, and other features. In WF, workflows can be defined using XAML [14][15][23]. XAML elements map directly to Common Language Runtime object instances, while XAML attributes map to Common Language Runtime properties and events on those objects. XAML files can be created and edited with visual design tools such as Microsoft Expression Blend, Microsoft Visual Studio, and the hostable Windows Workflow Foundation visual designer. They can also be created and edited with a standard text editor, a code editor such as XAMLPad, or a graphical editor such as Vectropy. Anything that is created or implemented in XAML can be expressed using a more traditional .NET language, such as C# or Visual Basic.NET. However, a key aspect of the technology is the reduced complexity needed for tools to process XAML, because it is based on XML. As a result, a variety of products are emerging, particularly in the WPF space, which create XAML-based applications. As XAML is simply based on XML, developers and designers are able to share and edit content freely amongst themselves without requiring compilation. As it is strongly linked to the .NET Framework 3.0 technologies, the only fully compliant implementation as of today is Microsoft's Environment [13][22][23].

Among all mark up languages XAML is best choice for developing device independent visual components. Since XAML is silverlight framework language with browser and operating system independent, this is correct and suitable technology for developing device independent visual components for web applications [14][21].

IV. IMPLEMENTATION

For a Silverlight Application this ability can be used so that regardless of how large or small you size the browser window, the Silverlight rendering engine will scale the user interface to fit while maintaining aspect ratio and functionality [14][15]. Note in these two screen shots how the user interface maintains layout, aspect ratio and functionality regardless of how large or small the browser window is sized:

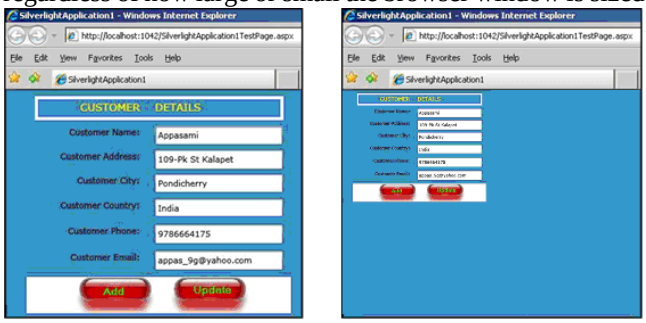

Figure 6. Device Dependent User Interface Controls.

Figure 7. Device Independent User Interface Controls.

In addition to the use of vector graphics, the application uses a bit of code to handle when the browser is resized and then scaling the outer canvas which contains all the application's user controls thus resizing the application to fit and maintain aspect ratio. The Figure 6 shows Device Dependent User Interface controls, but Figure 7 shows Device Independent User Interface controls. Here is how that is done.

A *ScaleTransform* is associated with the outer root Canvas:

```
<UserControl x:Class="SilverlightApplication1.SilverlightControl1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="400" Height="300">
   <Grid x:Name="LayoutRoot" Background="White">
     <Canvas x:Name="Root">
       <Canvas.RenderTransform>
         <ScaleTransform x:Name="ScalePage" ScaleX="1" ScaleY="1" />
       </Canvas.RenderTransform>
     </Canvas>
   </Grid>
</UserControl>
```

The code behind for the main page initializes the event handlers for resize events and registers the Page object as scriptable from the HTML page via JavaScript [21][22][23].

```
public Page()
{
    InitializeComponent();
    Application.Current.Host.Content.Resized
          += new EventHandler(OnContentResized);
    Application.Current.Host.Content.FullScreenChanged
          += new EventHandler(OnFullScreenChanged);
    HtmlPage.RegisterScriptableObject("Page", this);
    if (App.Current.Host.IsLoaded)
    {
        HtmlPage.Window.Invoke("ResizeSLContainer");
    }
}
```

The JavaScript function *ResizeSLContainer()* is then invoked. This function calls back into the Silverlight application to scale the canvas using the actual width and height of the Silverlight plug-in [14][15].

```
function ResizeSLContainer()
{
  var slPlugin = document.getElementById("SilverlightControl");
  slPlugin.Content.Page.ScaleContainer(slPlugin.content.actualHeight,
                                       slPlugin.content.actualWidth);
  if (slPlugin.content.actualHeight > 0){
     slPlugin.width = (slPlugin.content.actualHeight / 744) * 1005;
  }
}
```

Note the ability for the Silverlight application to invoke JavaScript and for the code behind to be invoked from JavaScript. The *ScaleContainer()* method is marked scriptable so that is can be called from JavaScript. This function performs the necessary math to set the *ScaleX* and *ScaleY* values on the *ScaleTransform* associated with the root Canvas [21][22][23].

```
[ScriptableMember]
public void ScaleContainer(double containerHeight,
                           double containerWidth)
{
    if (containerHeight != 0 && containerWidth != 0)
    {
      ScaleTransform scale =
        this.FindName("PageScale") as ScaleTransform;
      scale.ScaleX = containerHeight / 744;
      scale.ScaleY = containerHeight / 744;
    }
}
```

User Controls

A Silverlight Application is a User Control that is inserted into and rendered within the Silverlight Plug-In. Silverlight comes with a set of built in controls such as *Button*, *ListBox*, *DataGrid* and so on. As a Silverlight developer you also have the ability to define your own User Controls and embed them within other User Controls. This feature makes it very easy to divide the user interface functionality up into a set of reusable chunks that are organized using the built-in layout controls such as *Grid*, *StackPanel* and *Canvas* [16][17]. Figure 8 shows Margins for Silverlight controls. Here is how our sample application is laid out:

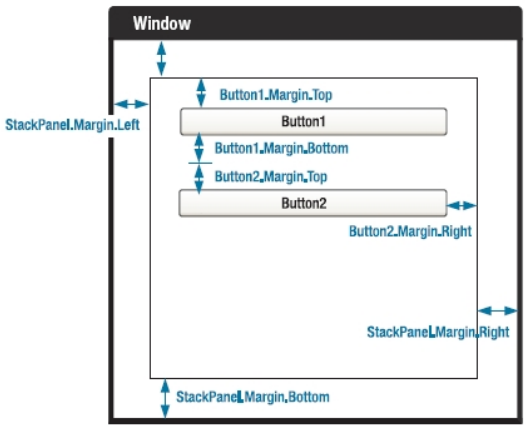

Figure 8. Margins for Silverlight controls

The outer most User Control is called Page. The XAML for Page contains the definition for the Banner and a reference to the User Control for the main application both of which are inserted into the root Canvas layout control [16][17].

```
<UserControl x:Class="SoundsFamiliar.Page"
    xmlns="http://schemas.microsoft.com/client/2007"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:my="clr-namespace:SoundsFamiliar"
    Width="1024"
    Height="768">
    <Canvas x:Name="Root">
        <Canvas.RenderTransform...>
        <!-- Banner -->
        <Canvas...>
        <!-- Application -->
        <my:MainWindow x:Name="MainWindow" Width="1024"
            Height="668" Canvas.Top="100" Canvas.Left="0"/>
    </Canvas>
</UserControl>
```

In order to embed a User Control into another User Control as we are doing here with *MainWindow*, we you must add an Xml namespace specification for the namespace where the User Control is defined [19][20][23].

V. CONCLUSION

Device independent visual components can be easily

developed using affine vector graphics and Silverlight. Since Silverlight is new technology in Dot net, it supports Browser independence and multi operating system. The Resolution Independent can be achieved by XAML language and Affine Vector Graphics. Since Silverlight and Affine vector graphics supports different resolution, different browsers and many operating systems, so we can easily develop device independent components Affine vector graphics and Silverlight.

VI. FURURE WORK

In future we can use Device independent visual components in devices like black berry, mobile, I-Pod and others. Single components for multi heterogeneous system support with same look and feel for all devices will be developed in future.

REFERENCES

[1] Cyril Concolato, J. Le Feuvre and J.-C. Moissinac," Design of an Efficient Scalable Vector Graphics Player for Constrained Devices", IEEE Transactions on Computer and Electronics, Vol. 54, No. 2, On page(s): 895-903, MAY 2008 .
[2] Liang Cheng, Jianya Gong, Xiaoxia Yang, Chong Fan, and Peng Han, "Robust Affine Invariant Feature Extraction for Image Matching", IEEE Geoscience And Remote Sensing Letters, Vol. 5, No. 2, April 2008.
[3] Sei Nagashima, Takafumi Aoki, Tatsuo Higuchit and Koji Kobayashi, "A Subpixel Image Matching Technique Using Phase-Only Correlation", IEEE 2006.
[4] P. K. Srivastava and B. G. Krishna, "Geometrical processing of Cartosat-1 satellite data," Bull. Nat. Natural Resour. Manag. Syst., no. (B)-30, pp. 7–16, 2005.
[5] Liang Cheng Jianya Gong Xiaoxia Yang Chong Fan Peng Han State Key Lab "Robust Affine Invariant Feature Extraction for Image Matching". Geoscience and Remote Sensing Letters, IEEE,,April 2008Volume: 5, Issue: 2, PP: 246-250
[6] R. Lukac, B. Smolka, K. Martin, K.N. Plataniotis, and A.N. Venetsanopoulos, "Vector filtering for color imaging", IEEE Signal Process. Mag. Spec. Issue on Color Image Process., 22, 74–86, 2005.
[7] Anna Derezinska and Tomasz Malek, "Experiences in Testing Automation of a Family of Functional- and GUI-similar Programs", International Journal of Computer Science & Applications, Technomathematics Research Foundation, Vol. 4, No. 1, pp. 13 – 26, June 2007.
[8] Honkala, M. Cesar, P. Vuorimaa, P,"A device independent XML user agent for multimedia terminals", Multimedia Software Engineering, 2004. Proceedings. IEEE Sixth International , On page(s): 116- 123, Dec. 2004
[9] Tai-Yeon Ku, Dong-Hwan Park, Kyeong-Deok Moon, "Device-independent markup language", IEEE Computer and Information Science, On page(s): 508- 512, Dec 2005.
[10] Abrams, M. Device-Independent Authoring with UIML. In W3C Workshop on Web Device Independent Authoring, Bristol, UK, October 3-4, 2000.
[11] Ali, M. F., M. Abrams, Simplifying Construction of Cross-Platform User Interfaces using UIML. January 2001, Paris, France.
[12] "Scalable Vector Graphics (SVG) Tiny 1.2 Specification", W3C Candidate Recommendation 10 August 2006.
[13] UIML2.0 Specification, http://www.uiml.org/specs/UIML2/
[14] Brad Dayley and Lisa DaNae Dayley, "Silverlight™ 2 Bible", Wiley Publishing, Inc., 2008.
[15] Matthew Donald, "Silverlight 2 Visual Essentials" Firstpress, 2008
[16] http://blogs.msdn.com/bobfamiliar/pages/resolution-independent-user-interface-user-controls-and-deep-zoom.aspx
[17] http://www.appleinsider.com/articles/06/12/21/apple_seeks_patent_on_resolution_independent_user_interface.html
[18] http://en.wikipedia.org/wiki/Resolution_independence
[19] http://developer.apple.com/releasenotes/GraphicsImaging/RN-ResolutionIndependentUI/
[20] http://chinese-school.netfirms.com/computer-article-device-independence-resolution.html
[21] http://www.silverlight.net
[22] http://code.msdn.microsoft.com/silverlightut
[23] http://silverlight.net/learn/tutorials/controls.aspx

Appasami. G was born in Pondicherry, India in 1980. He received his Master of Science degree in Mathematics and Master of Computer Applications degree from Pondicherry University, Pondicherry, India; Pursuing Master of Technology in Computer Science and Engineering from Department of Computer Science, Pondicherry University, Pondicherry, India.

Currently he is working as Research Assistant in Department of Computer Science, Pondicherry University, Pondicherry, India. His Area of interests includes image processing, neural network and web technology.

Suresh Joseph. K was born in Tamil Nadu, India in 1978 received his Bachelor of Engineering degree from Bharathiyar University and Master of Engineering (Computer Science and Engineering), from University of Madras, Chennai, India, in 2002.

Since 2006, he has been an Assistant Professor at the Department of Computer Science, Pondicherry University, and Pondicherry, India. His research interests include Image processing and Software Engineering.