8-3-2009

Microsoft Silverlight Photography Framework: Comparing Component Based Designs in Adobe Flex and Microsoft Silverlight

David Roossien
Grand Valley State University, david.roossien@dematic.com

Microsoft Silverlight Photography Framework
Comparing Component Based Designs in Adobe Flex and Microsoft Silverlight

David Roossien
August 3, 2009
Grand Valley State University
CS693 Masters Project
For Professor Robert Adams

Table of Contents

Introduction

Two similar photography framework projects were implemented in the new Rich Internet Applications (RIA's) Adobe Flex and Microsoft Silverlight. The first project was implemented during the Masters course called CS658 Distributed Computing. It was implemented using an Adobe Flex 3 client and an Action Message Framework (AMF) client/server interface. A PHP/MySQL server backend with a Zend Action Message Framework (AMF) interface was developed. A paper was written describing its features. The second project was implemented during the Masters course called CS693 Masters Project. It was implemented using Microsoft Silverlight 3 beta. This paper describes its features.

At the time of this project both Adobe Flex and Microsoft Silverlight were undergoing rapid development. At the beginning of this project Silverlight 2 had been released and Silverlight 3 was in beta. Silverlight 2's features were limited compared to Adobe Flex 3 so Silverlight 3 beta was used for this project. By the end of the project Silverlight 3 had been released and Adobe Flex 4 was in beta.

Flex and Silverlight provide large Application Programming Interfaces (API's). Applications can be designed using a variety of architectures. The purpose of this paper is not to perform a complete comparison between Flex and Silverlight. To do such a comparison would require massive amounts of time and research. The purpose of this paper is to describe the design and implementation of the Silverlight photography framework project. Along the way this paper compares the Silverlight project to the Adobe Flex project and discusses the goals of each product. During the comparison many important similarities and differences between Silverlight and Flex technologies are discussed.

Rich Internet Applications

RIA's have become extremely popular today for many reasons. For a list of features that are typical of RIA's see http://en.wikipedia.org/wiki/Rich_Internet_application. One of the key goals of all RIA's is to provide a more engaging interface so that users experience more of what the site has to offer without losing interest. Engagement is typically created through the use of a rich set of features such as animation, multimedia, user targeted content and user customizable content. The ultimate goal is to provide an experience that lures users back to the site, time after time, and encourages them to share their enthusiasm about the site with others. RIA's should provide such intuitive ease of use that its user do not give up or become frustrated with the limitations of the site.

Goals of Adobe Flex

In order to understand Flex's goals it is important to understand Adobe's goals for its suite of products. One of Adobe's goals is to leverage the vast number of designers who are already familiar with Adobe design tools.

"since Adobe technologies enable designers and developers to build RIAs with their current tools and skills, businesses can leverage existing personnel and assets to enhance customer engagement while minimizing expense."

(ref: http://www.adobe.com/resources/business/rich_internet_apps/benefits/#)

Adobe supplies a broad family of web and graphic design tools in its Creative Suite of applications. This includes Flash®, which has 98% usage (ref: http://en.wikipedia.org/wiki/Rich_Internet_application), Photoshop®, Illustrator®, and Fireworks®. Adobe's goal with Flex is to leverage Flash's dominance and familiarity with designers. It promotes the importance of a designer and developer workflow (ref: http://www.adobe.com/devnet/flex/workflow.html). For example, the output of a design from Fireworks can be imported and used as a template for a Flex application. Workflows make it easier to quickly translate designs into applications.

Adobe is focused on the business reasons for RIA development (ref: http://www.adobe.com/resources/business/rich_internet_apps/#). One of Adobe's main goals for Flex is to promote the rapid development of Flex applications. It hopes to make business ideas for applications easier to translate into applications. http://www.adobe.com/resources/business/rich_internet_apps/workflow/#

One of Flex's main goals is to provide a cohesive architecture that is transparent and familiar to designers so that designers and developers can readily communicate and quickly produce Flex applications. Flex provides easily understood solutions to very basic problems. These simple solutions require less training to learn and serve as building blocks for larger, more complex components. Flex lends itself to a rapid prototyping style of development.

Goals of Microsoft Silverlight

Microsoft promotes Silverlight as "the best of both worlds. By leveraging the .Net framework, our applications help you create visually rich experiences while providing backend support for rapid development." (ref: http://www.microsoft.com/silverlight/overview/ria/default.aspx)

One of Silverlight's goals is to leverage the Windows Presentation Foundation framework (WPF), the .NET Framework and its established developer base. WPF is used to provide a visual front end for all Windows based applications. WPF has access to the

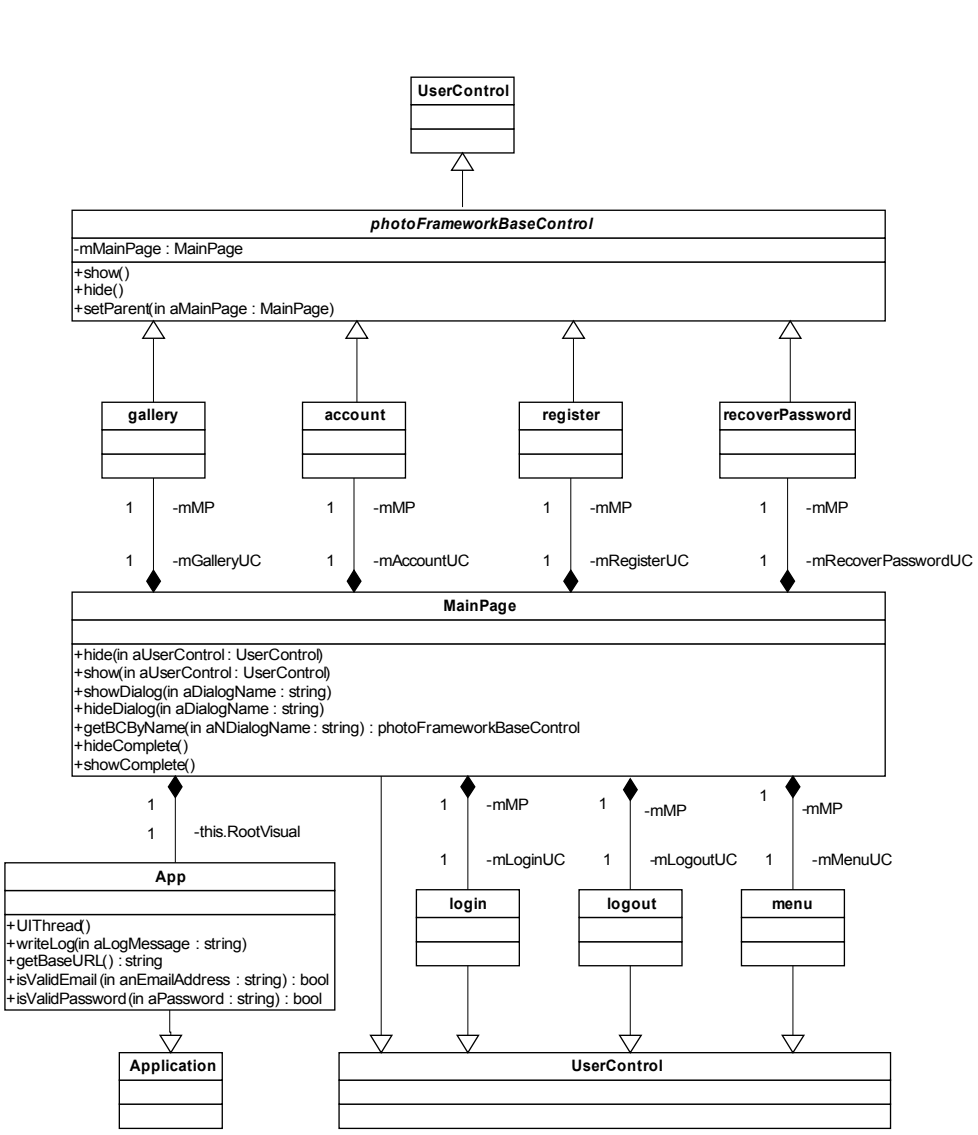
entire .NET Framework 3.5 class library, but Silverlight can only access a subset. (ref: http://msdn.microsoft.com/en-us/magazine/cc895632.aspx). While Microsoft promotes Silverlight's use of WPF as an advantage, some WPF developers are frustrated that Silverlight does not maintain many of the basic WPF conventions such as binding and event handling for all components (ref: http://silverlight.net/forums/p/12220/75006.aspx). Presumably, Silverlight strays from WPF when it needs to. There are additions to Silverlight that are not present in WPF. Consequently, it can be difficult to convert Silverlight applications into WPF applications and vice-versa. Regardless, Silverlight applications can still run on the desktop or in a web browser inside the Silverlight Media player. It is clear that one of Silverlight's goals is to allow WPF and .NET developers to develop Silverlight applications without much re-training.

Another of Silverlight's goals is to leverage the powerful Common Language Runtime (CLR, C# and VB.NET) and its high performance multi-threaded architecture. C# is a modern, strongly typed, object oriented language. It provides powerful programming constructs such as generics (templates), collections, events and delegates, inheritance and the conveniences of a modern programming language (ref: http://www.microsoft.com/silverlight/overview/ria/ria-details.aspx). One of Silverlight's distinguishing features is its support for multi-threading, which makes high performance multimedia based applications possible. Multi-threading can increase performance on multi-core processors. Silverlight makes use of multi-threading for heavy weight tasks such as asynchronous communication and animation. With this solid foundation it is clear that Silverlight is strong on architecture, performance and ready large scale RIA development.

Component Based Design

This section describes the Silverlight Photography Framework design. Figure 1 shows a UML Class Diagram with the major components and some of the important methods in the framework.

Figure 1 UML Class Diagram



App class

All Silverlight applications must inherit from the Application class. The App class inherits from the Application class and provides static methods including logging, validation and resources (including style) for the rest of the application. The App class has a MainPage component, which provides the visual content for the application.

MainPage class

The UserControl class is the basis for all the visual Silverlight controls. The MainPage class inherits from the Silverlight UserControl class. The MainPage class provides the visual content (framework) for the application. It provides methods for managing the visibility of the child components. It also maintains some basic state and global data for coordinating the information passed to its children. The MainPage component has several child component classes.

Child classes

Child component classes of the MainPage include the following:

- account
- changePassword
- gallery
- register
- recoverPassword
- login
- logout
- menu

The account, changePassword, gallery, register, and recoverPassword classes are modal dialogs that inherit from the abstract base control class, photoFrameworkBaseControl. The photoFrameworkBaseControl class inherits from the standard Silverlight UserControl class. The photoFrameworkBaseControl adds methods that all dialog components can use to manage visibility, ZIndex order, animations and a standard way to interact with the MainPage.

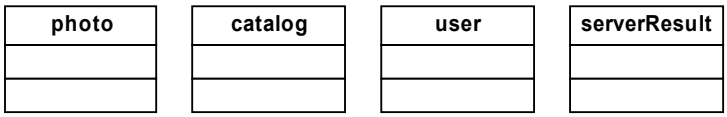
The menu, login and logout classes do not inherit from the photoFrameworkBaseControl. Instead, they inherit directly from the UserControl. A separate set of logic in MainPage was used to manage the visibility of the login and logout headers. There is never a need to hide the menu, therefore inheriting directly from the UserControl class was sufficient.

Data classes

The data classes used in this project are shown in Figure 2. These were used for object exchange with the AMF server gateway. The photo class contains all the available information about one photo. It includes things like title, description and the path to the image on the web server. For example, the gallery dialog maintains lists of photo objects that are displayed. It uses the photo data to load the images or to display the titles and descriptions. The catalog class provides pricing and print size information. The user class provides storage for user account information that can be returned from the server and displayed on the various dialogs. The MainPage class keeps the current user object and provides access for children controls to retrieve or modify the user object's data. After requesting the server to perform some action (i.e. log in) the serverResult object is returned to the client. The serverResult class provides a true or false return status plus a description. If the action failed it contains a description of why it failed. The cause of the failure can be displayed on screen used to assist the user with the desired action.

Figure 2 Data Classes

```
namespace photoFramework.classes
          (data classes)
```



Client and Server Communication

Figure 3 details the communication interface between the client and server. This design was also used in the Adobe Flex project, but is repeated here for easy reference. A client component sends a message to the gateway on the server. The gateway routes the message to the destination object. The object responds with a particular type of data. The data can be in the form of true/false, object or an array of objects. The protocol that was used for communication is the Action Message Format (AMF) as supported by Zend Framework 1.75.

Figure 3 Client/server communication diagram

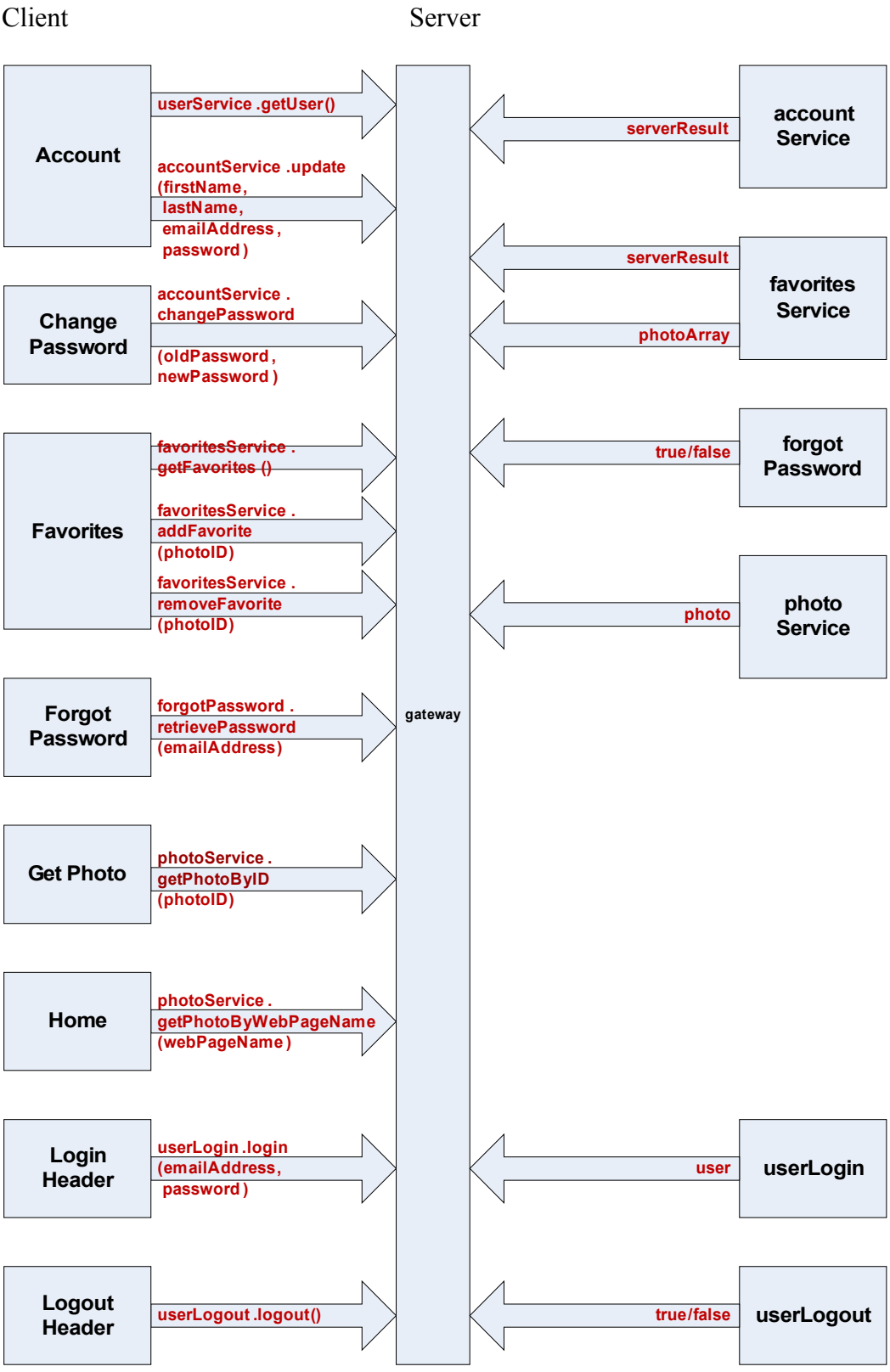
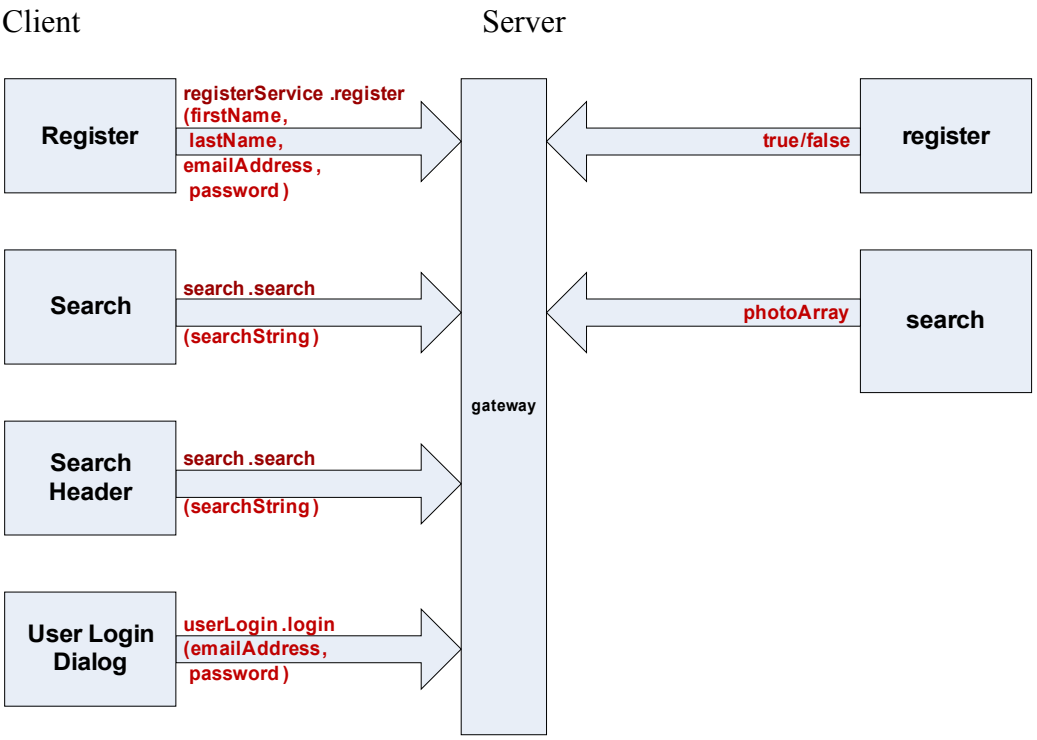


Figure 3 continued



Implementation

This section discusses many of the interesting details of the implementation. For comparison, it often references details of the Adobe Flex project. Both the Flex and Silverlight designs utilize a simple component based design with a main control component that manages the visibility of child controls. This provides an excellent basis for comparison of the Flex and Silverlight technologies. The only major client side differences are that the Flex project uses RemoteObjects for its client server communication, but the Silverlight client uses a custom AMF interface based on the FluorineFX library.

Components

The most basic building blocks of a component based design are the classes on which the components are based. Flex components are based on the UIComponent in the mx:containers namespace classes (ref: http://livedocs.adobe.com/flex/3/langref/mx/containers/package-detail.html). The UIComponent class inherits from a long list of classes including the DisplayObject and EventDispatcher. For example, the Flex canvas control was widely used to create Flex components (ref: http://livedocs.adobe.com/flex/3/langref/mx/containers/Canvas.html)

Silverlight components are based on the UserControl class, which is based on the ContainerControl class in the System.Windows.Forms namespace (ref: http://msdn.microsoft.com/en-us/library/system.windows.forms.usercontrol(VS.85).aspx).

During implementation it was discovered that there are important differences in the capabilities of the UserControl and UIComponent classes. One important difference is that the Silverlight UserControl is a lighter weight component and contains fewer useful properties and methods than Flex's UIComponent class. For example, the Flex UIComponent (DisplayObject) inherits from the EventDispatcher class (ref: http://livedocs.adobe.com/flex/2/langref/flash/events/EventDispatcher.html). Therefore, it has a large built in set of event handlers to choose from. In contrast, Silverlight's UserControl has very limited event handling. Since the UserControl contains fewer base methods and properties it follows that additional architecture is required to provide these features.

Visibility

Proper visibility management is a fundamental requirement in a component based design. In order to follow proper user interface design principles visibility changes should be coordinated at the parent level. It is common for animations to be activated when a

control is shown or hidden. It is also common for initialization to take place when a control is shown or hidden. Developers will need to design a system that provides the following features:

1. A component of the application, typically a menu or an event, can request that a child be shown or hidden.
2. The parent informs that the child that it is about to be shown or hidden. This gives the child the chance to initialize itself prior to becoming visible or hidden.
3. The parent either controls or activates the show/hide animations.

Visibility in Flex

In Flex, changing visibility is accomplished using a static reference to the DisplayObject. For example, in Flex, a programmer could write

```
Application.application.dialogName.visible = true;
```

This would make the dialog visible and, if it is already hidden, trigger the show() method inside the dialogName component (if one was implemented). For example, the following onShow() method would perform initialization on the dialog by clearing the fields in a form and then setting the focus to the first field in the form.

```
private function onShow():void
{
        clearFields();
        focusManager.setFocus(firstName);
}
```

Visibility in Silverlight

Silverlight user controls provide the property Visibility. The Visibility property can take on the following enumerated values:

- Visibility.Collapsed
- Visibility.Visible

To show a User Control set

```
this.Visibility = Visibility.Visible;
```

To hide a User Control set

```
this.Visibility = Visibility.Colapsed;
```

Unlike Flex, or its .NET UserControl counterpart (ref: http://msdn.microsoft.com/en-us/library/system.windows.forms.control.visiblechanged.aspx) which does support visibility events, the Silverlight UserControl does not provide an event that is called when a UserControl becomes visible or collapsed. Consequently, programmers will need to create their own events for inititializing the dialog.

During this project two different architectures were developed for managing the visibility of UserControls. One can be found in the MainPage class and it manages the visibility of the login and logout controls. A blog entry was written describing how to implement this method (ref: http://blog.davidroossien.com/?p=231). A second method can be found in the abstract class, photoFrameworkBaseControl. By inheriting from this base control, a dialog can utilize its built in visibility, animations and ZIndex management. A blog entry was written that describes how to implement this method (ref: http://blog.davidroossien.com/?p=259).

In a similar way to HTML, Silverlight uses a ZIndex. After making a UserControl visible it is necessary to temporarily increase the ZIndex of the UserControl. After hiding the control is may be necessary to lower it ZIndex so that it does not interfere with other controls.

Animations

Animations in Flex are created by using the mx:effects package (ref: http://livedocs.adobe.com/flex/2/langref/mx/effects/package-detail.html). Animations in Silverlight are created by using the System.Windows.Media.Animation package (ref: http://msdn.microsoft.com/en-us/library/system.windows.media.animation(VS.95).aspx) and in particular, the Storyboard class.

During this project it was discovered that Storyboards could not be bound to UserControls. This made it difficult for a parent to re-use the same Storyboard on multiple UserControl children. This problem was overcome by referring directly to each instance of the UserControl by its instance name. Unfortunately, this did not produce efficient code, but it did provide a workaround. Ultimately, this was overcome by creating a Storyboard for each UserControl and placing it in an abstract class, photoFrameworkBaseControl.

One other minor inconveneince was found with Storyboards. If they are interrupted during operation then an exception is thrown. Before triggering a storyboard it must first be stopped.

Developing an animation framework was easier in Flex than in Silverlight. This was mainly due to Flex's simple binding implementation. In Flex it was simple to manipulate the target, thereby, applying the animation to a different UIComponent. For example, the following defines a fade animation in Flex:

```
<mx:Fade id="fadeOut"  alphaFrom="0.3" alphaTo="0.0" target="{fadeTarget}"/
>
```

All that was required was to change the fadeTarget to a different UIComponent and start the animation. This made it simple to create a small and re-usable animation framework. Flex's binding allowed its animations to be re-used on different components. The animation could even be re-used against different types of objects.

Binding

Flex takes a simple approach to data binding. When a variable is declared in ActionScript it must be prefixed with [Bindable]. When the variable's value is referenced in MXML the curly brackets are required to tell the compiler that the variable was defined in ActionScript. After a change to the variable's value in ActionScript the control will be updated with the new value. (ref: http://www.adobe.com/devnet/flex/quickstart/using_data_binding/)

Silverlight's data binding is less intuitive than Flex and takes some getting used to. Binding in Silverlight can only be applied to something that inherits from the FrameworkElement class. It relies on something called the DataContext where a child inherits a bound variable from its parent. If an variable needs to be bound then it must be added to the DataContext of the parent. Once bound, if a variable's value changes then the child will receive the new value through the DataContext of the parent.

Examples provided in the Silverlight API showed that DataContext binding should be created in the constructor of a UserControl (ref: http://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext(VS.95).aspx). During implementation it was discovered that this did not work and was at best fragile. In order for the DataContext to produce binding it needed to be updated after the data was changed.

An example of Silverlight binding can be found in account.xaml and account.xaml.cs. For example, the user data class contains several member variables. The below XAML declarations show two different ways that a binding might be declared.

1. Binding a TextBox to User.mFirstName (long syntax)

```
<TextBox x:Name="firstName">
  <TextBox.Text>
     <Binding Mode="TwoWay" Path="MFirstName"/>
  </TextBox.Text>
</TextBox>
```

2. Binding a TextBox to User.mFirstName (short syntax)

```
<TextBox x:Name="lastName" Text="{Binding MLastName, Mode=TwoWay}"/
>
```

The following method was then used to change the value of the bound user object and update the DataContext.

```
// called when a new user object is received
public void setUser(user aUser)
{
   mUser = aUser;
   this.DataContext = mUser;
}
```

After executing the above statements the firstName TextBox's value would be changed to reflect the new value.

Validation and Forms

Validation is closely tied to the topic of binding. After a value changes an event that updates the binding could also be used to validate the data. Unfortunately, in Silverlight, there are no built in events that fire when a bound variable changes. Silverlight requires you to create a custom data class. Inside each of the setter methods an event must be manually created. An event handler must be provided and it can be used to validate the data. The custom data class must implement the INotifyPropertyChanged interface (ref: http://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged(VS.95).aspx , http://silverlight.net/blogs/jesseliberty/archive/2009/06/05/silverlight-validation-in-detail.aspx) Custom code must be written to implement the interface and to handle the validation inside the data class' setter. For example, this type of validation interface was implemented in the user data class.

Flex provides the mx.validators class for validating text or form data. The mx.validators class provides a convenient set of MXML based validators for common types of form data. These are attached to existing text fields in MXML. For example a string validator can be used to validate that a string is between x and y characters long. An email validator can be used to validate that the user has entered a valid email address. The below MXML demonstrates a String validator and an Email validator.

```
<mx:StringValidator source="{firstName}" property="text" minLength="2"
maxLength="20"/>
<mx:EmailValidator source="{emailAddress}" property="text"/>
```

Another convenient feature in Flex is that by default, binding provides an event and a default handler. The programmer can optionally implement the handler to provide custom validation. The default handler is called propertyChange and it handles an event of type PropertyChangeEvent. In Flex, the programmer can also specify a custom name for the handler using the simple syntax below.

```
[Bindable(event="fooChanged")]
public var foo:String;
```

(ref: http://livedocs.adobe.com/flex/3/html/help.html?content=databinding_8.html)

Flex provides the mx.containers.Form class for working with form data. The mx:Form class allows a great deal of work to be done in MXML, rather than in code. Combined with the mx.validators class, Flex provides an all MXML solution for designing and validating forms. Silverlight does not provide a XAML solution to form design and as previously discussed its validation is much more complex than Flex. Instead Silverlight requires the user to write individual textboxes and write custom data classes and event handlers to perform validation. An alternative to this brute force method is to use Silverlight 3's new DataForm control, which provides a large set of form handling features.

Fluorine Interface

The FluorineFX library (ref: http://www.fluorinefx.com/) provides an AMF interface (RPC) that allows a Silverlight client to communicate with an AMF server gateway.

The following code shows the creation of the network connection to the AMF server gateway. This is usually done in the constructor of the user control.

```
public account()
{
        // Required to initialize XAML
        InitializeComponent();

        // Create AMF net connection
        mNetConnection = new NetConnection();
        mNetConnection.ObjectEncoding = ObjectEncoding.AMF3;
        mNetConnection.NetStatus += new
NetStatusHandler(_netConnection_NetStatus);
            mNetConnection.Connect("http://localhost/photoFramework/gateway.php");

}
```

The below code shows a typical method used to call a method on the server using the Fluorine AMF interface.

```
private void getUser()
{

    mNetConnection.Call("userService.getUser", new userHandler(this), new object[]
{ mMP.MUser.MEmailAddress });

}
```

Inside the above call the first parameter specifies the class name, userService, and the method to execute, getUser (). A handler, userHandler, is provided so that the return data can be processed by the Silverlight component. Finally, a parameter, MEmailAddress, is passed inside the getUser() method. After a response is received from the server it is processed in the specified handler. Appendix A shows a typical handler with casting and hashmap parsing of the response.

The Fluorine AMF library proved not to be as simple to use as Remote Objects in Flex. Objects returned to Flex could be cast directly into data objects on the client. Instead, with Fluorine in Silverlight each object needed to be parsed as a hashmap of key/values where the values were of type string. There were no Fluorine examples that showed this procedure, but a method was developed to do the parsing. It was tedious work, but worth it because it allowed the server side PHP classes to be completely re-used.

Asynchronous Callbacks and Threading

During implementation, a difference in the thread model between Flex and Silverlight was discovered. Starting with Silverlight 2, Microsoft made a change so that asynchronous callbacks are received in a separate thread from the User Interface thread. When applying the Fluorine AMF interface a thread was created when the call was made. The thread lived until the result had been processed in the specified handler. A blog entry was written that describes the procedure for handling callbacks and calling the User Interface thread. It can be found here:

http://blog.davidroossien.com/?p=197

Unlike Silverlight, Flex callbacks allow the communication thread to access the UI thread asynchronously and directly, which can impact performance.

Performance

Since Flex callbacks return directly to the UI thread it can be interrupted when these asynchronous callbacks return. User Interface performance could degrade if there are many messages or a large volume of data being received while smaller messages are waiting to be processed. During testing, this performance degradation was observed in Flex animations. Animations stuttered after long lists of photo class objects were received by the callback inside the Flex application. Animations were also impacted by image downloads, which arrive asynchronously.

Asynchronous communication is of major importance in RIA applications. An advantage with Silverlight's approach is that the callbacks can be handled and managed concurrently outside of the UI thread. This allows for the possibility of managing the impact and/or timing of the interruption of the UI. It can possibly lead to smoother animations if it is managed properly.

Events

Silverlight's UserControl is roughly analogous to Flex's DisplayObject. Dialogs for each project were based on these basic classes. During implementation it was discovered that Flex has more useful events built into its DisplayObject class than are found in Silverlight's UserControl. This is a huge disadvantage for Silverlight development. Flex DisplayObject has built in the following events:

- CreationComplete
- Initialize
- Show
- Hide

Silverlight's UserControl class provides only the Loaded event, which is analogous to the CreationComplete event in Flex. It's hard to imagine why Silverlight doesn't include these very basic and useful events in the UserControl class. In what world would an application not use these types of events?

Event handlers

Flex events are more intuitive to implement than Silverlight, but Silverlight's approach has the potentially to be more powerful. Specifying an event in Flex's MXML allows great flexibility. Additional parameters can be attached to the event simply by adding them in the declaration and the event handler prototype. For example, the following

declares an image and its mouseDown event handler. The image passes not only the event but also an integer index to the ActionScript event handler dragIt().

```
<mx:Image id="p0" mouseDown="dragIt(event, 0)"/>

private function dragIt(event:MouseEvent, anIndex:uint):void
{
        var lImage:Image = event.currentTarget as Image;
}
```

The MouseEvent contains a pointer to the instance of the object that created the event.

In Silverlight, the standard event handler receives two arguments as follows

- object sender --the object that generated the event
- EventArgs e --contains no data when used with a standard event handler

Unfortunately, Silverlight does not allow additional parameters to be passed from XAML. It also does not provide a way for parameters to be added to the EventArgs. This can make it more difficult to accomplish some tasks. For example, the following XAML declares an event handler:

```
<Image x:Name="image0" MouseLeftButtonDown="changeMainPhoto"></Image>

private void changeMainPhoto(object sender, MouseButtonEventArgs e)
{

    Image lThumb = sender as Image; // cast

    String lName = lThumb.Name;
    String lSubString = lName.Substring(5);
    int lIndex = Convert.ToInt32(lSubString);

}
```

Passing an index is still possible, but the instance name needs to be manipulated to provide the information.

A second and possibly more powerful way to implement event handlers in Silverlight is to use the .NET Framework concept of a delegate. A delegate connects the event to its handler. Using the delegate allows custom data to be added to the EventArgs and a delegate method to be called to consume the data. For reference see the following link:

http://msdn.microsoft.com/en-us/library/system.eventhandler(VS.71).aspx

The delegate approach is potentially more powerful than the Flex approach because it allows a method from one type of class to be called by a class of an entirely different type. This allows for the development of specialized interfaces between classes that may not otherwise be able to communicate. For more information see the following reference:

http://www.c-sharpcorner.com/UploadFile/ddutta/EventHandlingInNetUsingCS11092005052726AM/EventHandlingInNetUsingCS.aspx

Resizing images

Rather than download several different sizes of the same image both RIA applications were designed to download the largest version of an image and resize it on the client. It was a fairly simple task to develop a static method for resizing images in a Flex client. See Appendix B for sample code. Resizing images in Silverlight was also simple. When declaring an image, simply specify MaxWidth and MaxHeight. In code, set the image source. Silverlight automatically resizes the image to the specified MaxHeight and MaxWidth. The following XAML and C# demonstrate resizing images.

```
<Image x:Name="image0" MaxHeight="100" MaxWidth="100"></Image>

// Code sets the source, but also resizes because the MaxHeight and
// MaxWidth properties are set
Image lImage = this.FindName("image0") as Image;

if (lImage != null)
{
        lImage.SetValue(Image.SourceProperty, lImageSource);
}
```

Drop down Menu

An important part of many applications is a drop down menu. Flex provides a drop down menu called the menubar control. It allows an XML configuration and is generally well documented. During the Flex project a blog entry was written detailing how to use the control (ref: http://blog.davidroossien.com/?p=94).

Unfortunately, Silverlight does not include a drop down menu component. After searching for a solution a suitable open source alternative was found at the below link:

http://www.nablasoft.com/guardian/index.php/2008/10/07/silverlight-dropdown-menu-control/

After prototyping with this menu control and fixing a few bugs it provided a rough replacement for the features of the Flex menubar. It did not support an XML configuration, but it turned out to be very useful. The drop down menu was added to the solution as an assembly (separate project) and it was referenced from the photo framework.

Style

Flex uses a traditional CSS approach. Components include a reference to the CSS using syntax like the following:

```
<mx:Style source="style/global.css" />
```

Each component can use a different style sheet. Components specify the style using traditional CSS syntax like the following:

```
.customLabel
{
        fontStyle: normal;
        fontFamily: Verdana;
        color: #245b95;
}
```

Elements can reference the style using styleName=”customLabel”. For example, a label would reference the .customLabel style using the following syntax:

```
<mx:Label x="42" y="111" text="Enter password:" width="140"
styleName="customLabel"/>
```

Silverlight takes a different approach to style. Styles are defined as Static Resources in App.xaml and compiled into the application. For example, the following defines a style called “customCanvas”:

```
<Application.Resources>
        <Style x:Key="customCanvas" TargetType="Canvas">
                <Setter Property="Background"
                        Value="#93A8BD">
                </Setter>
         </Style>
</Application.Resources>
```

User Controls reference the style using a StaticResource

```
<Canvas x:Name="loginHeaderCanvas"
        Style="{StaticResource customCanvas}"
        Width="400" Height="60">
```

TargetType specifies that a Canvas control will use the defined style. If there is a mismatch in TargetTypes at runtime, then an error will occur. Unfortunately, this additional type checking does not come with many benefits. The compiler will not catch a mismatch. As it loads each User Control during runtime this kind of mismatch renders the entire Silverlight application in an error state. For example, the user would see a dialog box with a message similar to the following:

> A first chance exception of type 'System.Windows.Markup.XamlParseException' occurred in System.Windows.dll
>
> Additional information: Cannot find a Resource with the Name/Key customCanvas [Line: 17 Position: 51]

Flex's approach does not catch mismatches, but at least it does not crash the runtime. Instead, mismatched style properties simply have no affect on the appearance of the control. Silverlight catches these mismatches and forces the programmer to correct the mistake.

MXML and XAML

Both Flex and Silverlight provide an XML based markup language. Both make it convenient for designing controls, but there are huge practical differences between Flex's MXML and Silverlight's XAML. Flex's MXML is compiled directly into ActionScript classes. Using Flex Builder, a developer can set a breakpoint directly in MXML and wait for it to be executed. After the breakpoint is reached the object and its property values can be inspected.

Silverlight's XAML is compiled into common runtime object code.

> http://en.wikipedia.org/wiki/Extensible_Application_Markup_Language

Unfortunately, this additional compilation step causes problems with debugging. During implementation it was not possible to debug runtime code in XAML. The following problems were discovered:

- Many mistakes in XAML were only revealed at runtime (or not at all)
- The debugger presented the exceptions in common runtime language code.
- No reference was given to the offending line in XAML

For example, a syntax error in MainPage.xaml would be caught during runtime and the debugger would open a file called MainPage.g.cs (common language runtime code). It often was not obvious where the mistake was made. XAML is a fine markup language, but it lacks a tightly coupled relationship with C# and the compiler. Clearly, MXML is better integrated with ActionScript and provides a superior debugging experience.

Development environment

Adobe provides Flex Builder for building Flex applications, which is based on Eclipse. Flex Builder provides an integrated design, development and debugging environment. Microsoft offers several development environment options for Silverlight—Expression Blend and Visual Studio and Eclipse (http://www.eclipse4sl.org/). Expression Blend provides design tools for creating XAML, but it does not let the programmer edit the code behind (filename.xaml.cs). Consequently, Expression Blend does not let the programmer debug applications. Visual Studio 2008 provides an environment for editing filename.XAML and writing the code behind filename.XAML.cs. However, Visual Studio 2008 only provides a preview of the XAML design. Consequently, it does not allow the developer to select components and manipulate them. For example, unlike Flex Builder, it does not allow a developer to select a component and then highlight it in code. During this project the Eclipse for Silverlight environment was not used because it is relatively new.

Analysis: Flex and/or Silverlight

One of the goals of this project was to create a component based photography framework based on Microsoft Silverlight 3 beta. This provided an opportunity to explore the inner workings of Silverlight and compare it to a similar project that was implemented in Adobe Flex 3.

It was discovered that Silverlight makes a component based design more difficult to implement, but it can potentially produce higher performance user interfaces than Flex. Silverlight provides separate threads for asynchronous callbacks. It also takes advantage of threading for animations. This provides the potential for applications with more asynchronous communication, smoother response and higher performance user interfaces. Client machines with multi-core CPUs will notice smoother animations while the client loads images in other threads running on the other core(s). In Flex, all user interface activities take place in a single thread. This can result in stuttering animations and lower performance when receiving a large volume of asynchronous messages such as retrieving images from the web.

Silverlight requires more forethought, infrastructure development, object oriented design, and a solid understanding of its architecture before projects can be implemented. The

reward for this added expense is the ability to create large, monolithic, web based applications that still maintain high performance. For example, Silverlight might provide a stronger platform for 3D animation than Flex. A properly designed, multi-threaded Silverlight client has the potential to process large volumes of data without impacting its visual performance.

During this project it was found that many features that can be taken for granted in Flex are more difficult to implement in Silverlight and require additional development time. A few of these are:

- Lack of show/hide events built into components and general Visibility management complexity
- More complex binding implementation
- Lack of a menubar control
- No drag and drop manager built in
- More complex form validation
- Development environment shortcomings, code behind files
- More complex event handling system
- More complex asynchronous callback threading (but possibly higher performance)

Currently, one of the more pressing issues facing Silverlight is the development environment. So far, none of the Microsoft environment allows the programmer to graphically edit, write code and debug at the same time. Instead, Microsoft would have its users switch to Expression Blend to design an application, but code and debug in Visual Studio. Flex Builder has all of this built in. It also takes longer to use Silverlight's code behind files (i.e. filename.xaml + filename.xaml.cs). Switching back and forth just takes longer. Flex allows the ActionScript code to be located inside the MXML file. When combined with the XAML debugging problems described in a previous section, Flex just provides faster development.

New users will find that Flex is easier to learn than Silverlight. Flex solves the little problems for users, like providing a menubar control, drag and drop handling, a simple data binding implementation and built in show/hide event handlers in all of its DisplayObjects. Flex provides smaller building blocks from which users can create larger controls. This promotes incremental development and faster learning.

In contrast, Silverlight leaves many simple user interface tasks unfinished. It allows developers freedom to architect solutions to these smaller problems and instead, provides larger, monolithic components than Flex (i.e. the data form control). Larger controls can be powerful, but are also more complex and difficult to learn. When you think about the goals of Flex it makes sense that Flex is more focused on presentation, the needs of designers, rapid development, and the business reasons for RIA development. Microsoft is more focused on high performance, flexible architecture and developers who are already familiar with the .NET and WPF frameworks.

Both Flex and Silverlight are open source so controls can be customized. Below are links to the current open source sites:

- Flex: http://opensource.adobe.com/wiki/display/flexsdk/Flex+SDK;jsessionid=5655D8C0A9D40A09EBA7A01A1AF2EFB9)
- Silverlight: http://www.microsoft.com/downloads/details.aspx?FamilyID=2050e580-f1d5-4040-bb09-e6185591b6b5&displaylang=en

During both projects an active open source community was helpful in supplying components to solve certain problems.

Conclusions

After implementing both projects it is clear that the goals of Silverlight are significantly different than Flex. Silverlight's architecture is less compact, and ironically more flexible than Flex. As a result, more architectural pieces need to be assembled to accomplish the same tasks in Silverlight. Silverlight's UserControl contains fewer important methods, events and properties than Flex's UIComponent. This forces developers to add these features, but it provides an opportunity to provide a higher performance design that is tailored to the application. Though it may change, at this time, Silverlight is not a platform for rapid application development. It is missing too many basic features and leaves too many basic user interface problems unsolved. Instead, it is an application programming interface that can be used to develop components with the features you need.

Flex has lower startup costs than Silverlight in terms of infrastructure. This is mainly due to Flex's compact architecture with extensive features built into controls. Many common user interface problems are solved for developers and this makes developing Flex applications intuitive and fast. Flex's goals are aligned with designers and users of its design tools. Flex facilitates lower cost, incremental and rapid RIA development, which makes it easier to sell ideas to decision makers. The costs of developing Flex applications are easier to measure and predict than with Silverlight. Therefore, it is easier to make business decisions about future projects based on Flex than with Silverlight. To say that smaller applications are probably better done in Flex would be an oversimplification, but as applications grow in size and complexity Silverlight becomes a more compelling option. Silverlight's goals are aligned with software engineers who understand that an investment in architecture can yield huge performance benefits. Silverlight provides a platform for building higher performance applications than Flex due to its support for threading. This is an important consideration when designing an application and choosing a RIA platform.

Appendix A Return processing using the Fluorine Interface

```
/*
public void ResultReceived(IPendingServiceCall call)
{

    IDictionary openWith = call.Result as IDictionary;

    photoFramework.classes.user lUser = new user();

    foreach (DictionaryEntry de in openWith)
    {

        if (de.Key.ToString() == "emailAddress")
        {
            lUser.MEmailAddress = de.Value as String;
            App.writeLog("account userHandler ResultReceived() key = " +
de.Key.ToString() + ", de.Value = " + de.Value as String);
        }
        else if (de.Key.ToString() == "firstName")
        {
            lUser.MFirstName = de.Value as String;
            App.writeLog("account userHandler ResultReceived() key = " +
de.Key.ToString() + ", de.Value = " + de.Value as String);
        }
        else if (de.Key.ToString() == "lastName")
        {
            lUser.MLastName = de.Value as String;
            App.writeLog("account userHandler ResultReceived() key = " +
de.Key.ToString() + ", de.Value = " + de.Value as String);
        }
        else
        {
            App.writeLog("loginHeader logoutHandler ResultReceived() ERROR: unknown key =
" + de.Key.ToString());
        }

    }

    App.UIThread.Dispatcher.BeginInvoke(() => _account.setUser(lUser));

}
```

Appendix B Resizing Images in Flex

```
// (ActionScript)
public function resizePhoto(anImage:Image, aMaxDimension:uint):void
{
	var lRatioOrig:Number;

	lRatioOrig = anImage.width / anImage.height;

	// If width > height
	if(lRatioOrig > 1.00)
	{
		anImage.width = aMaxDimension;
		anImage.height = aMaxDimension * 1/lRatioOrig;
	}
	else
	{
		anImage.width = aMaxDimension * lRatioOrig;
		anImage.height = aMaxDimension;
	}


}
```