FaceOut Overview

A Silverlight 2 Line-of-Business Application

This document describes the features and technical design of the exemplar code-named “FaceOut”. This application is a line of business application built using the Silverlight 2.0 platform.

Todd Snyder
11/7/2008

Contents

Table of Figures

Introduction

FaceOut is a line of business application that allows a sales person to track their customer order history and other key performance indicators (KPIs). The application was built using the Silverlight 2.0 platform, Microsoft Live SDK, and the October CTP release of *NetAdvantage for Silverlight Data Visualization*.

Windows Communication Foundation (WCF) services are used to access application data – two XML files based on the NorthWind database. Customer information and KPI data is stored in these two files.

The application was built over the course of several small sprints using a pragmatic SCRUM approach. Each sprint focused on building out a different set of features.

Because of the rich data binding support built into the Silverlight platform, we used the concepts of the Model View View-Model (MVVM) Pattern to separate the concerns of each feature across several classes, separating the UI logic and supporting classes from the underlying data access and logic.

To see FaceOut in action visit: http://labs.infragistics.com/silverlight/faceout/

Application Overview

Figure 1 – Main Screen of the FaceOut Application

FaceOut uses a familiar dashboard approach to layout, breaking the main UI into different logical modules, each of whose content is based upon the selected customer. Behind the scenes, these modules are defined using UserControls, one for each module in the UI: My Customers, Order History, Key customer KPI(s), Customer & System Notes, and Customer Location. As a customer is selected from the *My Customers* list, the content of the other modules is updated.

My Customers

This module displays the list of contacts (customers) for a sales person. When a contact is selected, the other components on the screen are updated to display the corresponding customer information for the selected contact.

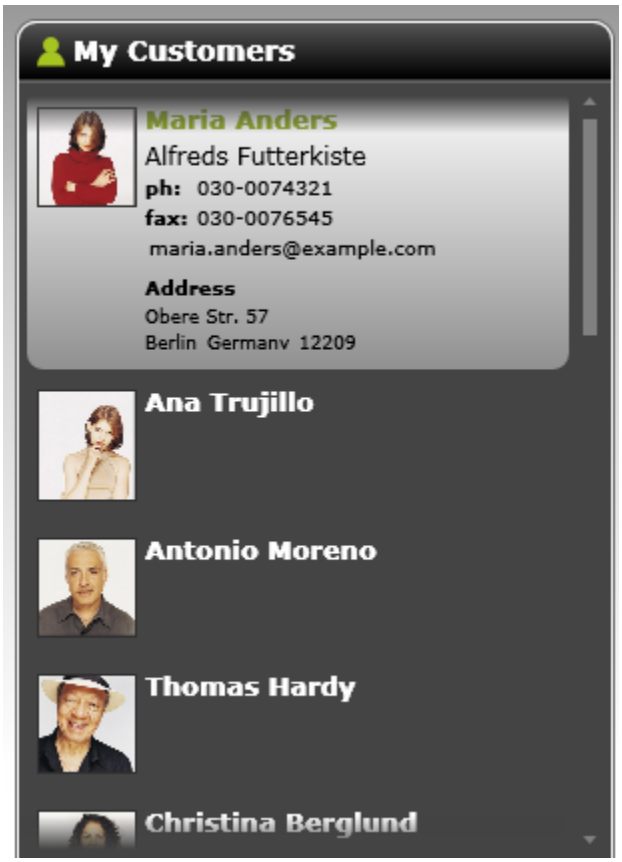


Figure 2 – My Customer's List Box

Order History

This module displays the last twelve months of order history for the selected customer. The total sales for each month are calculated and displayed using the xamWebChart's "Column" ChartType.

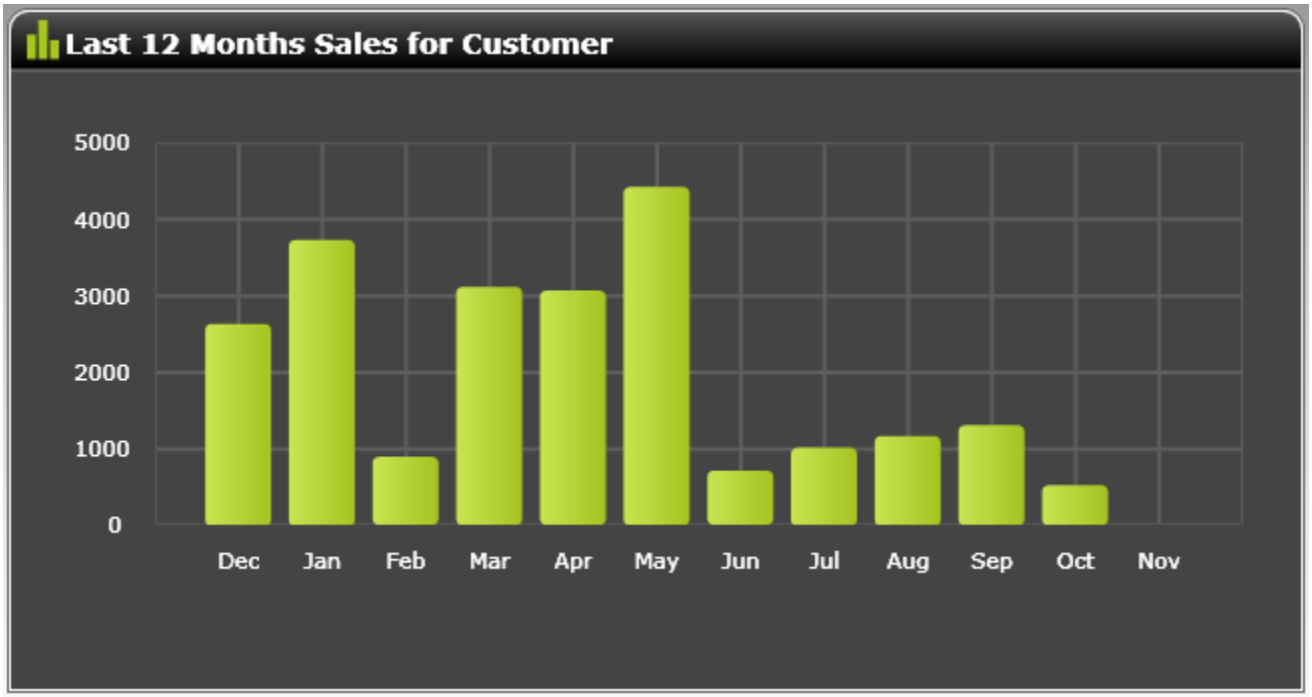


Figure 3 – Order History displayed using the xamWebChart Control

Customer Key Performance Indicators

This module uses the xamWebGauge to show the total number of sales the selected customer has made for the current period. The vertical black bar represents the target sales goal while the green fill indicates actual sales. The right gauge indicates the percentage of overall sales this customer has made in relation to the total sales team.

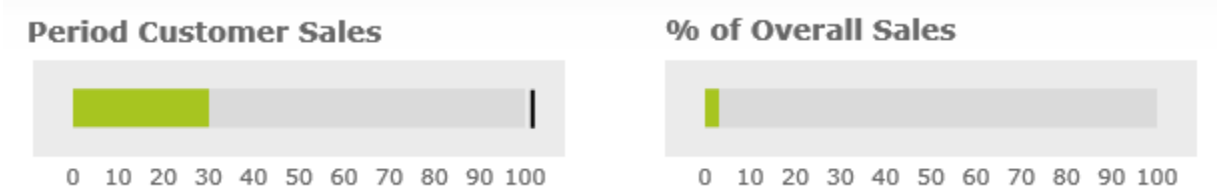


Figure 4 – KPI(s) displayed using the xamWebGauge Control.

Customer & System Notes

This module displays a call log of the communication between the sales person and their target customers. Additionally, it displays the important dates for the last major sales transactions.

Figure 5 – Customer call log and system notes

Customer Location

This module displays the primary location for a customer using the Microsoft LIVE Map control. As you select different customers, the map is updated with their location. Hover over the pushpin to see the address for the current customer. The control supports all the features of the LIVE SDK, letting you switch between 2D/3D, Road, Aerial, or Bird's Eye view.

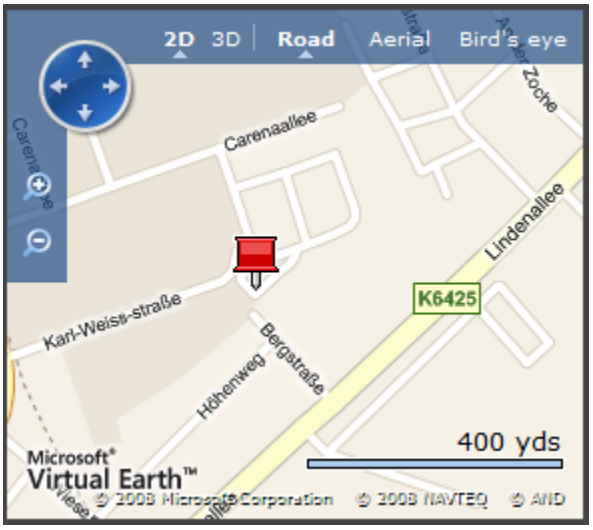


Figure 6 – The Live Map Control displaying a customer location.

Technical Notes

- FaceOut was built using the Silverlight 2.0 and .Net 3.5 platforms. The application uses Windows Communication Foundation (WCF), LINQ, and the *Net Advantage for Silverlight Data Visualization* components to display customer and sales order information.
- When integrating HTML content, the HTML content (div) must always have a z-index higher than the Silverlight plug-in's z-index.. The HTML content must be hidden (Visibility = Collapsed) in order to overlay any Silverlight content.
- XML files were used for the data source to simplify the distribution of the application. For production systems a database server such as SQL Server 2005 or later should be used instead.

Technical Design

This section presents an overview of platforms, frameworks, controls and design patterns used when building FaceOut.

Key Technologies

FaceOut is built upon the following technologies and toolsets:

- Silverlight 2.0
- .Net 3.5 (Windows Communication Foundation and LINQ)
- NetAdvantage for Silverlight Data Visualization (Oct 2008 CTP)
- Windows 2008 Server

NetAdvantage for Silverlight Data Visualization (October 2008 – CTP)

The application takes advantage of the data presentation and visualization features of the *NetAdvantage for Silverlight Data Visualization* control suite. The following controls included in this suite are used to display customer KPIs for a sales person: xamWebChart and xamWebGauge.

xamWebChart

The xamWebChart allows you to take advance of the rich data visualization capabilities of the Silverlight platform to display KPIs. Here, the "Column" ChartType of the xamWebChart is used to display the last 12 months of sales for the selected user. As the user changes, and thus the data behind the chart changes, the column chart transitions smoothly to the new values. This built in animation makes the transition from user-to-user feel more seamless and like a story rather than like an abrupt change of thought. See link for more details about the control.

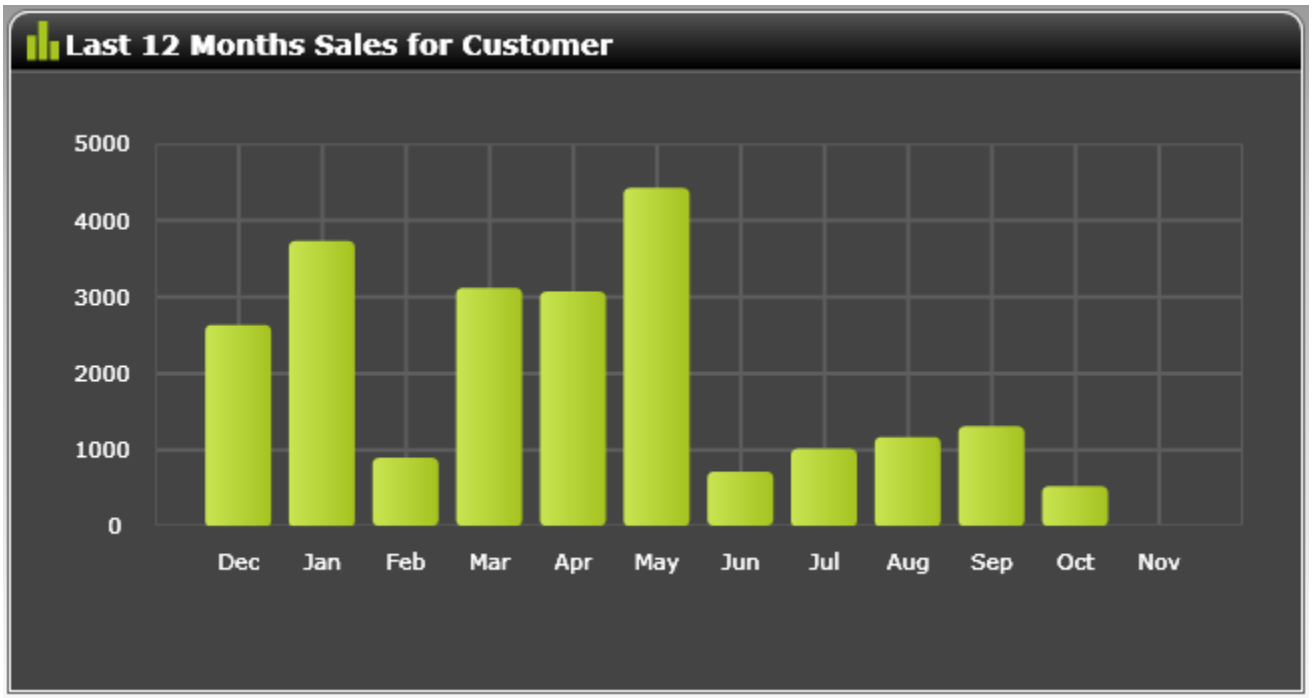


Figure 7 – xamWebChart displaying order history

xamWebGauge

Two instances of the xamWebGauge are used to display sales performance data in a "glanceable" format. You can quickly tell how well a user is performing against their own goals and as a percentage of the overall sales team. This xamWebGauge configuration is just one of many that can be achieved with this highly flexible control. Visit the NetAdvantage for Silverlight Data Visualization Feature Browser to see the xamWebGauge gallery and a number of additional samples. See link for more details about the control.

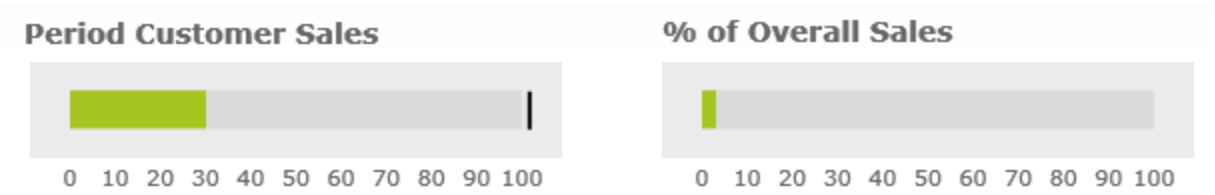


Figure 8 – xamWebGauge displaying KPI(s)

Live SDK Integration

The Microsoft LIVE SDK provides a rich platform for building web-based mashups that allow you to integrate the components of the LIVE platform into your applications. FaceOut integrates the Live Map control to display a customer's location. Because of Silverlight's built in support for integrating with a browser's DOM it was easy to create an HTML host control and custom JavaScript helper class to interface with the LIVE Map control.

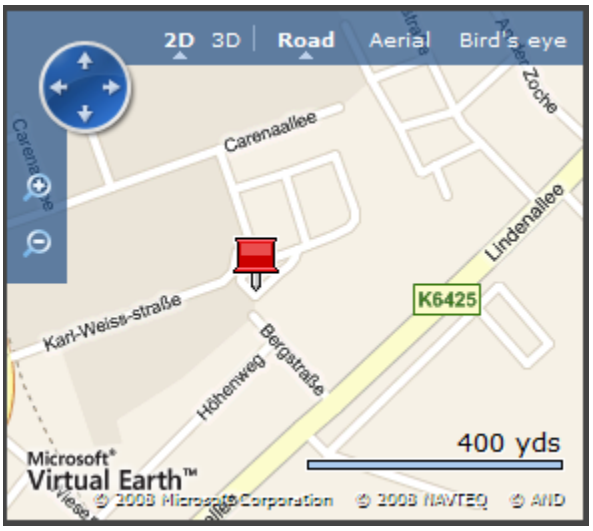


Figure 9 – Live Map control displaying a customer's location

Windows 2008 Server

This application is deployed on a production environment that is running Windows 2008 Server and Internet Information Services (IIS) 7.0. The hosting ASP.NET application and WCF services used by the application are set up as virtual servers under IIS. However, Windows 2008 Server is not a requirement for running the application -- it can be run using the Visual Studio 2008 Cassini web server or any machine running IIS 6.0+. See the following link to learn more about Windows 2008 and IIS 7.0.

Layers

This section gives a quick overview of the logical view (Layers) of the application. The application was built using the Silverlight 2.0 platform and uses Windows Communication Foundation (WCF) services to access its customer data store.

The application has three logical tiers:

- Client (Silverlight plugin running in a web browser)
- Middle Tier (WCF Services used for accessing the data store)
- Data Store (XML files used for storing contact and customer sales information)

<Add logical Picture>

Design Patterns and Principals

The key to successfully building a quality enterprise-based system is to utilize proven best practices and design approaches. While building the application we utilized several industry best practices for building enterprise .Net applications. This section outlines the key design patterns and design principals we followed to build FaceOut.

Principals

- Separation of Concerns: This is a principal where you separate the concerns for a use case into several different classes. For example, the Model View View-Model pattern separates code into three class types: View, View-Model, and Model.
- Single responsibility principal: This is a principal where you focus on making classes have high cohesion and try to avoid overloading the reasonability of a class.
- Refactoring: This is a design technique for restructuring an existing body of code, altering its internal structure without changing its external behavior.
- Request-Reply Message Pattern: This pattern uses a one-way communication channel. The service request is packaged as a single request object and the service result is returned as a single result object. Inside of each object (request/result) there may be one or more child objects.

Model View View-Model Pattern

The Model View View-Model (MVVM) pattern is a variation of the Model View Controller/Model View Presenter pattern. The pattern promotes separating the concern for a use case across several classes. Because of the rich data-binding support in Silverlight, the pattern is ideal for building applications on this platform.

Distinct class types in the MVVM pattern:

- View: Includes UI components and controls. These classes are responsible for only rendering the data for the application.
- View Model: A set of classes that is responsible for abstracting away the business layer (Domain Model) of the system, responding to user actions (e.g. Save button clicked), and maintaining the state of the view (e.g. which item is selected).
- Model: A set of classes that is responsible for representing the real world domain of the application and enforcing the business rules and logic of the application.

Reference:

http://martinfowler.com/eaaDev/PresentationModel.html

How the application uses the pattern:

Unlike Windows Presentation Foundation (WPF) the Silverlight 2.0 platform does not yet have full support for the MVVM pattern. The major piece missing in the Silverlight platform is support for RoutedCommands (`ICommand` interface). RoutedCommands make it easier to define the relationship between a view and a command using just XAML.

Because Silverlight does not have support for RoutedCommands yet, we had to roll our own implementation for FaceOut. To keep things simple, we implemented basic commanding support in FaceOut. All the major actions of the application (Load Selected Customer, Load Contacts, Etc…) are defined as separate commands that inherit from the `IViewCommand` Interface. We use the Interface to provide a consistent set of behaviors across all commands.

Each command implements the `Execute` method and is called from one of the view's code behind classes. The view is only responsible for providing the command with the necessary parameters (e.g. selected customer) and the command handles calling the business services and updating the view model/domain model with the data returned from the service.

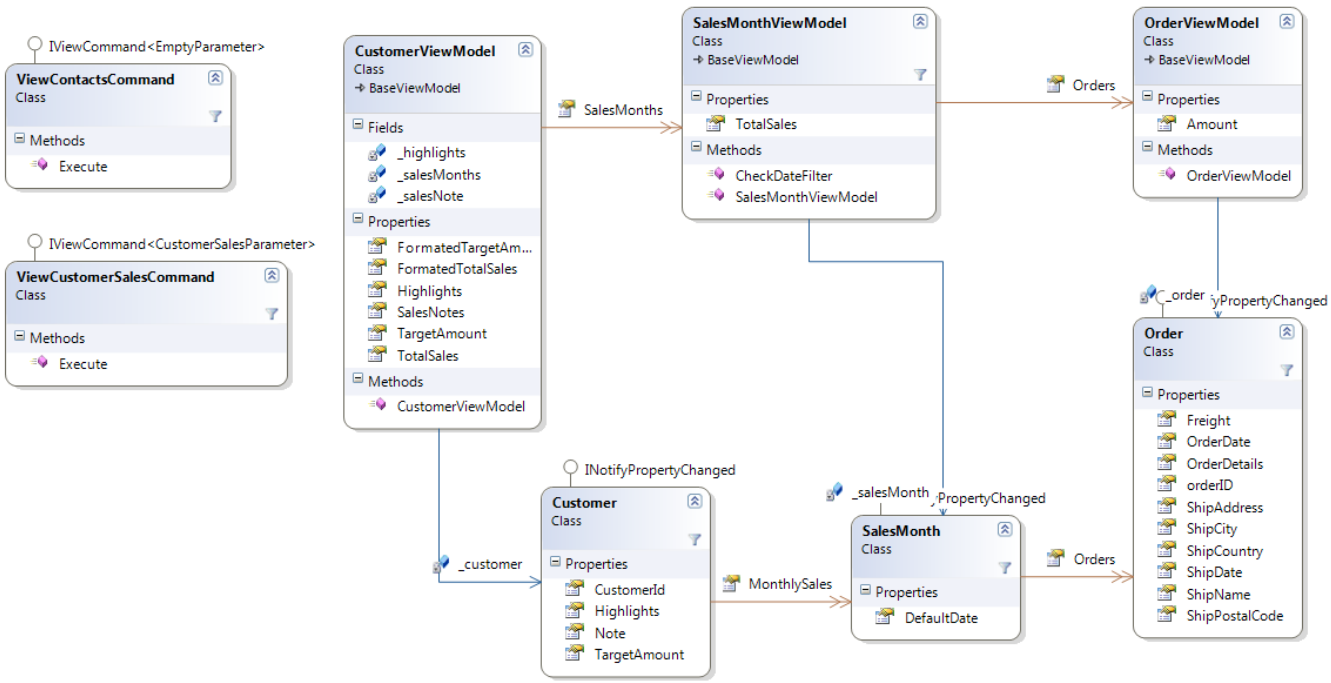
The application's `DataContext` is bound to a custom `UserContext` class that is responsible for being the top-level class in the view model layer. This approach makes it easy to abstract away the underlying structure of the domain model from the view and provides a consistent entry point into the view model.

Because of the simplicity of the FaceOut application, the domain model is used as Data Transport Objects (DTO) that are passed between the Silverlight client application and the WCF business services. In a more advanced

scenario (Complex domain model), you may need to have a separate set of DTO classes that are used between the client/service layer and use the domain model purely as a business layer.

The diagram below displays the relationship between the core view model and domain model classes used in FaceOut. The command classes are responsible for handling user requests (Load Contacts, Load Selected Customer, Etc…), calling the business service layer, and instantiating the view model.

The view model tracks the current state of the view and applies an abstraction over the domain model. The view model is responsible for implementing the `INotifyPropertyChanged` interface and creating any `DependencyProperties` the view wants to bind to.



Repository Pattern

The Repository Pattern is a data access pattern that promotes loosely coupling the domain model for a solution from its data sources. The main goal of the pattern is to act as an in-memory data repository that uses DataMapper-type classes to load data into and out of the domain model.

By using this approach you can achieve a higher level of persist ignorance that will increase the overall testability of your code. Further, you can easily handle adding additional data sources or changing the data access technology (e.g. from ADO.Net to LINQ or NHibernate) you use without affecting the core of your system.

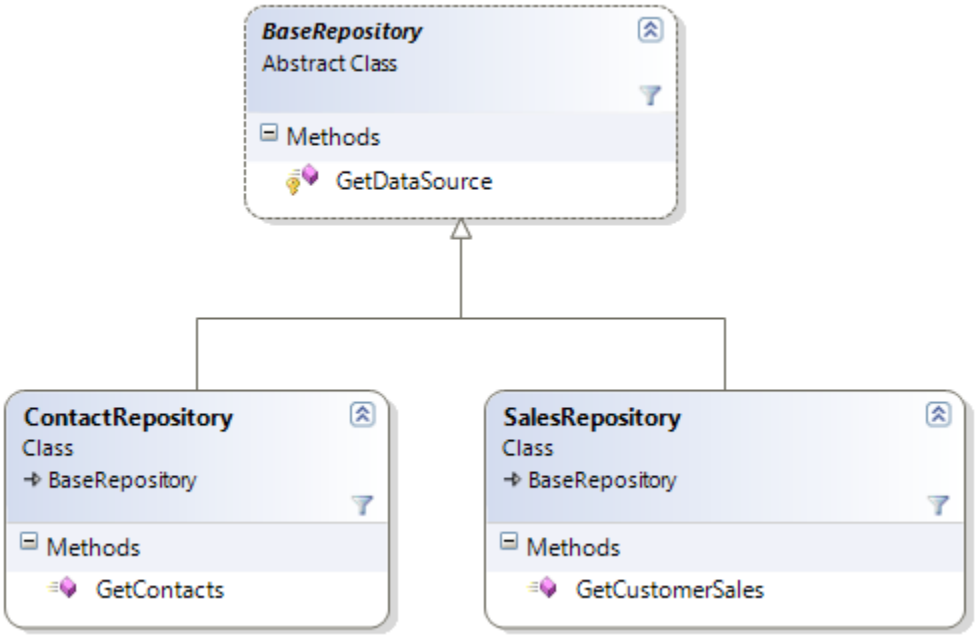
Reference:

http://martinfowler.com/eaaCatalog/repository.html

How the application uses the pattern:

The diagram & code listing below illustrates how the Repository Pattern is used by FaceOut. Because most of the data access is “read only” we went with a simple Repository implementation that maps the XML data source(s) into our domain model using XLINQ.

The `BaseRepository` is used to load the XML Data source, and each `BaseRepository` sub class (`ContactRepository` or `SalesRepository`) is responsible for mapping the data to the domain model using XLINQ.



Summary

The FaceOut exemplar demonstrates how to build upon the technologies supported by the Silverlight 2.0 platform to deliver enterprise-level applications.. It uses backend WCF web services to access the customer information displayed in the application. This document outlined the approach we took for building the application as well as how we were able to utilize *NetAdvantage for Silverlight Data Visualization* controls to build an engaging dashboard-style application on the Silverlight 2.0 platform.