PDSA.com

Solutions for the Real World

PDSA Special Report

The Importance of Services

Everyone is talking about Service Oriented Architecture (SOA) these days. However, this is nothing new if you have been doing good OOP design and programming all along. A service is nothing more than a class or a set of classes that provide you with some functionality that can reused from application to application or even within the same application. For example, if you create a class to wrap up retrieving a connection string from your application's configuration file, you have created a connection string service.

While most architects and developers think SOA equates to some kind of Web Services, the truth is SOA can be implemented using any kind of remoting service. SOA can even be implemented by running all services locally on the same computer. Even if all you do is separate classes into different assemblies and reference those assemblies from your main application, you are still doing SOA, it is just done logically instead of physically.

A Bad Programming Example

Let's take a look at a bad programming example where a developer might write code within their Web UI layer to populate a DataGrid. The following code shows some very bad two-tier code.

```
public void ProductsLoad()
{
  DataTable dt = new DataTable();
  System.Data.SqlClient.SqlDataAdapter da;
  string sql;
  string conn;

  sql = "SELECT * FROM Products";
  conn = "Data Source=Localhost;Intial
        Catalog=Northwind;Integrated Security=Yes";
  da = new System.Data.SqlClient.SqlDataAdapter(
          sql, conn);

  da.Fill(dt);
  grdProducts.ItemsSource = dt;
}
```

The above code should be self-explanatory. You are loading a DataTable object with data retrieved from a SQL Server database through a SqlDataAdapter object. The major problem with this code is if you need to retrieve data from the Products table in another type of UI, you will end up duplicating this code as shown in Figure 1.

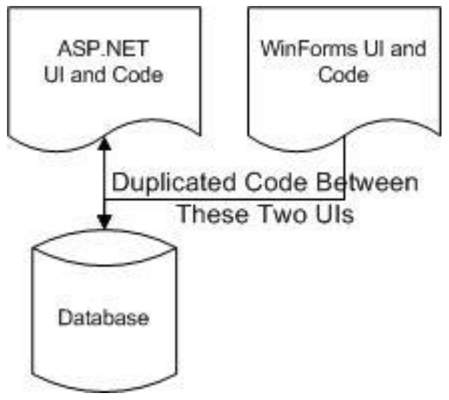

Figure 1: Two-tier applications suffer from a lot of redundant code.

There are quite a few things wrong with the code in the ProductsLoad procedure shown above, namely:

- SqlClient classes should not be referenced in the UI layer.
    - Makes it difficult to switch to a new provider.
- SQL statements should not be in the UI.
    - Can't reuse SQL, can't optimize it without recompiling code.
- Connect String should be in a Config file, the Registry, XML file, etc.
    - Hard coding makes switching databases difficult.
- No exception handling.
    - Need some error reporting other than default .NET exception messages.
- No reuse of code possible between different forms/applications.

To rewrite this code so it can be reused does not take a lot of work, simply creating a few extra classes with methods is all that is needed.

A Good Programming Example

If you were to break the bad programming example into the appropriate classes for a minimum service oriented architecture, you would need the following classes.

- AppConfig class
- WebException class
- Products Business Class
- Products Data Class
- Data Layer Class

To create a truly reusable Framework you would need a few more base classes from which you can inherit from. Figure 2 shows an example of the set of classes you might create to get a robust, enterprise class framework that can handle any programming job.

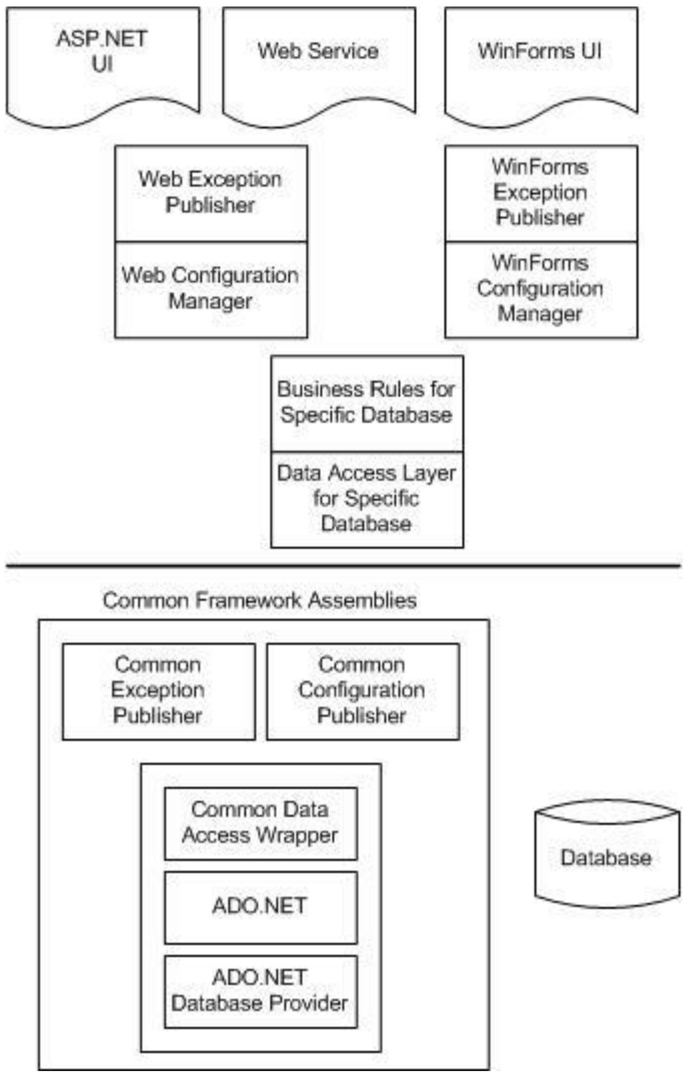
ASP.NET UI
Web Service
WinForms UI
Web Exception Publisher
Web Configuration Manager
WinForms Exception Publisher
WinForms Configuration Manager
Business Rules for Specific Database
Data Access Layer for Specific Database
Common Framework Assemblies
Common Exception Publisher
Common Configuration Publisher
Common Data Access Wrapper
ADO.NET
ADO.NET Database Provider
Database

Figure 2: A good SOA Framework will make your coding go much faster and provide extensive re-useability.

By using this approach you can build reusable services (classes) that can be called from different forms within the same application, from different types of applications (web, windows, web services, etc.)

So to re-write the bad programming example into a good programming example, the final code would end up looking like the following:

```
public void ProductsLoad()
```

```
{
  ProductManager mgr = null;

  try {
    mgr = New ProductManager();

    grdProducts.DataSource =
       mgr.GetProducts(AppConfig.ConnectString);
  }
  catch (Exception ex) {
      WebException.Publish(ex);
  }
}
```

This UI code is now just the bare minimum needed to get data and feed it to the appropriate UI control for display. Notice there are no connection strings, SQL, specific data providers, or anything else that is not reusable. The Products class is responsible for connection strings, SQL and the specific database provider. The connection string comes from a static method on the AppConfig class. All exceptions are published via the Publish method on a class called WebException. Looking at this code you can understand it without having to get bogged down in details. This is an important acid test of how well your code is separated into services. If you can understand the logic in the UI without seeing a myriad of details, you know you have achieved your goal.

The actual mechanics of separating out each of these services should be fairly easy to grasp. In fact, you will find this type of separation explained in many articles published in magazines and on the Internet.

Summary

Implementing a good OOP architecture is important to creating reusable code that will stand the test of time. By separating your code into multiple classes that provide a set of services to your applications, you will accomplish this goal. The PDSA .NET Productivity Framework (www.pdsaframework.com) uses all of these techniques. So, if you do not wish to create this Framework yourself, contact us for your free demonstration on how you can save hundreds of hours and thousands of dollars by purchasing our Framework.

Contact Information

If you would like to know more about the information in this special report, please contact either Paul D. Sheriff or Michael Krasowski at PDSA.

Paul Sheriff

(615) 675-4632

PSheriff@pdsa.com

Michael Krasowski

(714) 734-9792 x223

Michaelk@pdsa.com

Company Information

PDSA, Inc.
17852 17th Street
Suite 205
Tustin, CA 92780

Tel (714) 734-9792
Fax (714) 734-9793
www.pdsa.com

PDSA.com
Solutions for the Real World