

1 Background

We discussed the code-generator-generator approach (cstone.exe) in the first part of the white paper, and followed it up with a somewhat detailed discussion of the “pickled-object” code pattern in the second part.

In this part, we will examine extensions of the pattern, leveraging some of the functional characteristics of C# 1.0 to offer a “remote-object” pattern called Azure¹.

This pattern is dependent on the functionality provided by the BrightSword Reminisce™ Web Server, which exposes a completely RESTful interface to a database using and extending the Olive patterns.

Discussion of the Reminisce WebServer™ is outside the scope of this section, but is recommended as prerequisite reading.

The “remote-object” or Azure pattern allows for a standard WinForms or otherwise traditional “thick-client” application to bind to and utilize objects from a web-server *without web services*.

Before the user dismisses this approach as inconsequential, or worse, counter-productive, it is instructive to note that the ideological schism between proponents of the RESTful and Web-Service Oriented philosophies still exists – primarily because of valid counter-arguments from both sides as to the architectural validity of each approach.

Our take on the matter is that while there is valid reason to use Web Services in many scenarios, there are also reasons to use REST-based simple object access mechanisms in many other scenarios. There is also a severe paucity of tools and platforms available to the developer for the latter approach, which results in some pretty ugly shoe-horning of Web Services tools and methods to achieve the ends which could more elegantly be achieved with a RESTful approach.

2 Synopsis

BrightSword LongReach™ includes a set of pre-developed patterns which can be used to effect the “remote-object” pattern using a RESTful web interface and a logical extension of the “pickled-object” approach.

The Azure extensions and patterns are provided with the default installation of olive.exe, so no new tools are required to generate the azure patterns.

3 Technical Discussion

3.1 Summary of the RESTful approach

The RESTful approach, very simply put, is that every object in a system is given an unique resource identifier (URI), and operated on using the primitives of the HTTP protocol specification.

This means that with a RESTful web server, an object is mapped to an URL, and operations on the object can be mapped to the HTTP verbs of GET, POST and DELETE.

¹ The name Azure is a somewhat cheeky reference to the Microsoft's All-Powerful Object-Remoting platform code-named Indigo. Azure is indeed a very light shade of Indigo. Sorry Michael, the humor is regretted. ☺

Our implementation of a RESTful server uses a superset of the following convention to deal with objects and sets of objects.

- A GET request on the URL http://server/app/<object_type>/ returns an XML document containing the set of all known objects of type <object_type>
- A GET request on the URL http://server/app/<object_type>/<object_guid>/ returns an XML document containing the single object of type <object_type> identified by <object_guid>
- A POST request on the URL http://server/app/<object_type>/<object_guid>/ creates or updates the object of type <object_type> identified by <object_type>
- A DELETE request on the URL http://server/app/<object_type>/<object_guid>/ deletes the object of type <object_type> identified by <object_type>

Other primitives exist in Reminisce™, but are not of serious consequence to the rest of our discussion.

3.2 Application of the Olive patterns to the RESTful approach

Using the XmlLoader pattern on the client and the equivalent SqlServer(Fast)XmlLoader pattern on the server, it is possible to instantiate a set of objects on the client from an XML document generated on the server.

Therefore, if there is a request made from a client program to the server for an URL of the form {GET, http://server/app/<object_type>/} or {GET, http://server/app/<object_type>/<object_guid>/}, the XML document returned can be reconstituted into traditional objects.

Further, using the FormSerializer pattern on the client and equivalent FormLoader pattern on the server, it is possible to post a request to the URL of the form {POST, http://server/app/<object_type>/<object_guid>/} to effect the creation or update of an object on the server.

A library is available which encapsulates the transport binding of the URL to the mechanisms provided in the System.Net namespace, and effect transparent use of objects.

The key point to be noted is that the server is unaware of the client's operations of consuming the XML for the purpose of reconstituting objects, or posting the contents of the object in a way identical with that of a traditional browser, so no new functionality needs to be installed on the server to perform these operations.

The sample code given below is working code from an actual project.

The AzureLoader is modeled exactly after the traditional SqlServerObjectLoader discussed in Part 2 of the white paper, and supports identical semantics of lazy-loading.

The AzureStorageAdapter is modeled exactly after the traditional SqlServerStorageAdapter discussed in Part 2 of the white paper, and performs Persist and Delete operations. Erase is not supported by design, as there is no HTTP verb associated with ERASE.

```
private static void TestAzureGet()
{
    BrightSword.Azure.Core.Connection cxn = new BrightSword.Azure.Core.Connection();
    cxn.Credentials = System.Net.CredentialCache.DefaultCredentials;

    System.DateTime dtStart = DateTime.Now;

    CaseCollection Collection = CaseAzureLoader.Load(cxn, new QueryParameters(), new CaseCollection());
    System.DateTime dtFinish = DateTime.Now;

    foreach(Case c in Collection)
    {
        System.Console.WriteLine("{0} created at {1}", c.ID, c.Internal_CreatedTime);
        foreach (Comment co in c.Comments)
        {
            System.Console.WriteLine("\tComment: {0}", co.Text);
        }
    }

    TimeSpan ts = dtFinish - dtStart;
    System.Console.WriteLine("Retrieved {0} records via azure in {1} ms.", Collection.Count, ts.TotalMilliseconds);
}
```

Figure 1: Accessing Objects on the Client via the AzureLoader

```
private static void TestAzurePut()
{
    const int cRecords = 10;
    for (int i = 0; i < cRecords; i++)
    {
        try
        {
            BrightSword.Azure.Core.Connection cxn = new BrightSword.Azure.Core.Connection();
            cxn.Credentials = System.Net.CredentialCache.DefaultCredentials;

            Case c = new Case();
            c.BillableLineCount = 23;
            c.Code = "Jazz Man";

            Comment co = new Comment();
            co.Text = "This is REALLY cool!";
            c.Comments.Add(co);

            c.Persist(CaseAzureStorageAdapter.PersistFunc, cxn);
        }
        catch (Exception ex)
        {
            System.Console.WriteLine("Exception [{0}] while creating a record {1} via azure.", ex.Message, i);
        }

        System.Console.Write(".");
    }

    System.Console.WriteLine();
}

private void TestAzureClear()
{
    BrightSword.Azure.Core.Connection cxn = new BrightSword.Azure.Core.Connection();
    cxn.Credentials = System.Net.CredentialCache.DefaultCredentials;

    CaseCollection Cases = Case.GetObjects(new QueryParameters(), new CaseAzureLoader(cxn));

    Cases.Delete(new StorageFunctor(CaseAzureStorageAdapter.DeleteFunc, cxn));

    return;
}
```

Figure 2: Saving and Deleting Objects on the Client via the AzureStorageAdapter

4 Description of Suite Components

4.1 *olive.exe*

When the LongReach™ suite is installed, the MSI installs the code-patterns for the AzureLoader and AzureStorageAdapter, allowing for direct generation of the Azure patterns. A runtime link to the BrightSword Azure library is required for execution.

No further configuration is required – even permissions traditionally required by XmlSerializer are done away with.

5 Summary

The “remote-object” pattern, in conjunction with a RESTful web server, is an elegant and powerful way to develop rich-client applications interacting with an object storage server over HTTP.

Since the interaction with the server is over simple (S)HTTP, there is **no separate server component or server configuration required**. The very components that serve the objects up to a browser and accept posted forms are used to interface to the Azure client.

The critical distinction between the Web Service approach and this approach is that Web Services require significant processing on the server and client to marshal function arguments and results back and forth. This would be reasonable when the RPC functionality afforded by the Web Service approach is required, but in cases where no RPC is required, Azure forms an extremely lightweight and efficient approach to effecting object-remoting.

The Azure patterns have been used in production, where they have given satisfaction to the developer (in making it trivial to build distributed applications), and to the customer (in providing a seamless, rich user-interaction experience with minimal configuration issues).

6 Credits

The idea stems from the work of Dr. Roy Fielding, whose seminal paper is hereby cited² as the guiding principle of this approach.

Credit is also given to my friend and colleague Jim Baker, whose approach to building a RESTful web server in python, was the inspiration to developing the Reminisce™ Web Server.

² <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>