Meta Resource Management System

Technology Evaluation

René Freude (707168) <rene.freude@gmx.de>
Daniel Sadilek (707297) <sadilek@tfh-berlin.de>
Stephan Weiß (706830) <steph.weiss@web.de>

2003-09-21

Revision History

Revision 0.1	2003-09-21
	Initial public release.

This document contains the insights we obtained while evaluating the technology we will use for the implementation of the MRMS.

Table of Contents

1. Overview

We will implement a three-tier architecture using the .NET Framework. For this, we have to select a database, a runtime environement for the server and an interprocess communication implementation for the communication between client and server. For development, we need an Integrated Development Environment (IDE) and a testing framwork.

2. Development Environement

We will use Visual Studio .NET as IDE. We made all tests with it: It is easy to use and comprehensive; we only missed functions for refactoring and testing. For the latter we found a a .NET implementation of the xUnit approach: NUnit (http://www.nunit.org), which is nicely integrated into Visual Studio .NET: It can be run from within it and you can jump with one click to failed tests. Writing tests is as easy as with JUnit, but instead of naming conventions NUnit uses .NET attributes to mark test classes and methods:

```
using System;
using NUnit.Framework;
using System.Diagnostics;

namespace Tests
{
  [TestFixture]
  public class CSharpTest
```

```
  {
    [SetUp] public void SetUp()
    {
    }

    [TearDown] public void TearDown()
    {
    }

    [Test] public void TestMe()
    {
      Assertion.Assert("failing test", false);
    }

    [Test] public void TestTrace()
    {
      // Trace/Debug messages won't be output when running all
      // tests in assembly.
      Trace.WriteLine("Trace...");
      Trace.Assert(true == false, "true is not false",
        "- more detail");
      Trace.WriteLine("Hello World!", "Trace");
    }
  }
}
```

3. Database

The .NET Framework provides ADO.NET as the standard data access technology. It has a strong abstraction layer (compared to Java-JDBC), leaving us the flexibility of choosing a suitable layer of abstraction.

Our initial intention was to use a native XML database for persistent object storage. When we tried to find a free production stable XML database, it showed up, that such thing is not available. A further evaluation of free OODBMS products brought very similar results. Therefore we decided to fall back to a relational database solution combined with object relation mapping. The only free OR mapping implementation we found does not support transactions, so that we will have to implement our own OR mapping solution.

The necessary standard database access is provided by the ADO.NET library, that is shipped with the .NET runtime environment. We found one driver written for MySQL and one for PostgreSql. Development of the latter has just been started, so we decided to use MySQL as database server.

3.1. Evaluation Code

The following code tests a basic database query and illustrates how MySQL can be used with a .NET application. The top level object needed first is a connection object (class `MySQLConnection`), that is created with a connection string holding the needed information to access the MySQL database server. After opening the connection (`connection.Open()`), a command object (class `MySQLCommand`) can be created, that encapsulates the preparation and execution of a SQL command. Navigation through the results of a query can easily be done with the returned object of type `IDataReader`.

```
using System;
using System.Data;
using NUnit.Framework;
using ByteFX.Data.MySqlClient;

namespace Tests {
    [TestFixture]
    public class MySQLTest {
        // the connection object
        private IDbConnection _connection;

        [SetUp]
        public void SetUp() {
            // prepare connection string
            string conString  = "Server=locahost;Database=test;"
                              + "User ID=dba;Password=sql";
            // create connection object with connection string
            _connection = new MySqlConnection(conString);
```

```
            // open database connection
            _connection.Open();
        }

        [Test]
        public void testCommandExecution() {
            IDbCommand command;
            // prepare command
            command = _connection.CreateCommand();
            command.CommandText = "SELECT firstname, lastname "
                                  + "FROM employee";
            // execute command and get reader
            IDataReader reader = command.ExecuteReader();
            while(reader.Read())
            {
                string FirstName = (string) reader["firstname"];
                string LastName = (string) reader["lastname"];
                Console.WriteLine("Name: " +
                    FirstName + " " + LastName);
            }
            reader.Close();
            command.Dispose();
        }

        [TearDown]
        public void TearDown() {
            _connection.Close();
        }
    }
}
```

4. Communication between Client and Server

The .NET Framwork provides two means for the communication between a client and a server:

1. Web Service

   In .NET a web service is implemented as a .asmx file which must be deployed into an installation of Microsoft's Internet Information Server (IIS) that supports ASP.NET.

2. .NET Remoting

   “The .NET Framework provides a number of services such as activation and lifetime control, as well as communication channels responsible for transporting messages to and from remote applications. Formatters are used to encode and decode the messages before they are sent along a channel.” [MSDN] There are different channels distributed with the .NET Framework: TcpChannel which is based on TCP-Sockets and HttpChannel which is based on the well-known HTTP; also there are different formatters: BinaryFormatter for performance critical applications and SoapFormatter where interoperability with other remoting systems is essential.

So, it seems as if you have the ability to create web services by using .NET Remoting? Yes, and to make the situation completely unmanageable you may host your .NET Remoting application either in the IIS or stand-alone.

The following table lists the differences between .NET Remoting and web services.

Table 1. .NET Remoting vs. Web Services

.NET Remoting	Web Services
Faster when using TcpChannel and BinaryFormatter	Slower
May run inside IIS and stand-alone	Need an IIS
When stand-alone: Server runs on every Windows version >= 95	IIS is only available for the server versions of Windows

.NET Remoting	Web Services
Supports events	Do not support events; client must poll changes on the server
Server objects may have a global state or a state per client or no state at all	Must be stateless
Supports remote references	Every object is passed by value
Lower interoperability with other systems and/or programming models	Higher interoperability
Full support for the .NET type system, perfect type fidelity	Support for only a subset of .NET's types

From this table you can draw the conclusion that .NET Remoting is much more powerful while web services are more interoperable. Since interoperability is not a key-requirement for us because we want to implement the client with .NET, as well, we have chosen to use .NET Remoting. However, the following section where we present some code examples that show how to use .NET remoting also contains Java code that accesses the .NET server process via SOAP.

4.1. Evaluation Code

4.1.1. .NET Service

To write a class that can be used with .NET Remoting the only thing you have to do is to implement the marker interface `MarshalByRefObject`.

```
using System;

namespace remotingTypes {
  public class MyService : MarshalByRefObject {
    public MyService() {
      Console.WriteLine("Instance of MyService created.");
    }

    ~MyService() {
      Console.WriteLine("Instance of MyService deleted.");
    }

    public DateTime getServerTime() {
      Console.WriteLine("getServerTime() invoked");
      return DateTime.Now;
    }

    public MyType getMyType() {
      return new MyType("Hello Java!", 1234);
    }

    public int add(int x, int y) {
      return x + y;
    }
  }
}
```

4.1.2. .NET Type

The service above contains a method named `getMyType()` which returns an object of the type `MyType`. It is shown in the following listing.

```
using System;

namespace remotingTypes {
  [Serializable] public class MyType { // passed by value
  // public class MyType : MarshalByRefObject { // passed by reference
    public string content;
    public int number;

    public MyType(string content, int number) {
```

```
      this.content = content;
      this.number = number;
    }

    public string getTestContent() {
      return content;
    }

    public void remoteMethod() {
      Console.WriteLine("remoteMethod() invoked.");
    }
  }
}
```

4.1.3. .NET Server

To make the service available we need to have a server process that opens a channel, registers the type `MyService` as a server remoting type and keeps running until the user the stops it (in our example by pressing enter). Then, the service is reachable via the URI `http://localhost:8082/object1uri` and its Web Service Description Language (WSDL) file can be downloaded from `http://localhost:8082/object1uri?wsdl`.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using remotingTypes;

namespace remotingServer {

  class ServerProcess {
    [STAThread]
    static void Main(string[] args) {
      IChannel channel = new HttpChannel(8082);
      ChannelServices.RegisterChannel(channel);

      MyService object1 = new MyService();
      ObjRef refl = RemotingServices.Marshal(object1, "object1uri");
      Console.WriteLine("URI: " + refl.URI);

      Console.WriteLine("Press enter to end.");
      Console.ReadLine();

      RemotingServices.Disconnect(object1);
      ChannelServices.UnregisterChannel(channel);

      GC.Collect();
      GC.WaitForPendingFinalizers();
    }
  }
}
```

4.1.4. .NET Client

We wrote a little tester that connects to the server process and invokes some methods. The call of `object1.getMyType().remoteMethod()` which prints out the string "remoteMethod() invoked." shows the difference between parameters and return values passed by reference and passed by value. If `MyType` is declared as "`[Serializable] public class MyType`", `object1.getMyType()` returns an object of type `MyType` by value and the string is printed out on the client. If `MyType` is declared as "`public class MyType : MarshalByRefObject`", `object1.getMyType()` returns a remote reference to an object on the server and the string is printed out on the server.

We tested the performance of the HttpChannel and the SoapFormatter by making 1000 successive calls: It took less than five seconds, thus it took less than five milliseconds per call which we consider to be fast enough.

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using remotingTypes;
```

```
namespace remotingClient {
  class ClientProcess {
    [STAThread]
    static void Main(string[] args) {
      IChannel channel = new HttpChannel();
      ChannelServices.RegisterChannel(channel);

      RemotingConfiguration.RegisterWellKnownClientType(
        typeof(MyService), "http://localhost:8082/object1uri");

      MyService object1 = new MyService();
      if (object1 == null) {
        Console.WriteLine("Could not locate server.");
        return;
      }

      Console.WriteLine("ServerTime: " + object1.getServerTime());
      object1.getMyType().remoteMethod();

      int x = 0;
      DateTime start = DateTime.UtcNow;
      for (int i = 0; i < 1000; ++i) {
        x = object1.add(x, i);
      }
      Console.WriteLine("x: " + x);
      DateTime end = DateTime.UtcNow;
      Console.WriteLine("Time: " + (end-start).ToString());

      Console.ReadLine();
    }
  }
}
```

4.1.5. Java Client

We also wanted to check whether we could connect to the .NET server process via Java: We downloaded the WSDL file and used the WSDL2Java tool from the Axis project (http://ws.apache.org/axis/) to create Java stubs from it. It worked, even only a little bit slower than the .NET client (the 1000 calls took 6.7 seconds), but: the parameters and return values can only be passed by value and their types may only contain public properties (no methods).

```
public class Tester {
  public static void main(String[] args)
    throws MalformedURLException, ServiceException, RemoteException {
    MyServiceService service = new MyServiceServiceLocator();

    // Now use the service to get a stub which implements the SDI.
    MyServicePortType port = service.getMyServicePort();

    System.out.println("Got port.");

    System.out.println("Server time: " + port.getServerTime());
    MyType myType = port.getMyType();
    System.out.println("Content: " + myType.getContent());
    System.out.println("Number: " + myType.getNumber());
    System.out.println("3 + 5 = " + port.add(3, 5));

    int x = 0;
    long start = System.currentTimeMillis();
    for (int i = 0; i < 1000; ++i) {
      x = port.add(x, i);
    }
    System.out.println("x: " + x);
    long end = System.currentTimeMillis();
    System.out.println("Time [ms]: " + (end - start));
  }
}
```

4.1.6. Lessons Learned

.NET Remoting is powerful, fast and easy to use. It is interoperable with Java to a certain degree: If we only use simple data container types that act like structs we can use the service with Java.

5. Server Runtime Environment

The server implements the core functionality of our resource management system and must provide it to client applications on the network. The decision for a runtime environment depends on the technology we use to expose functionality to clients on the net. As a result of the evaluation of communication technologies, we decided to use .NET Remoting instead of Web Services. One deciding point was, that .NET Remoting works stand-alone with the standard runtime environment, while Web Services - written in ASP.NET - need to be run inside a web server. No free and suitable substitute for Microsoft's Internet Information Server (MS-IIS) could be found, so switched to .NET Remoting and will only need the .NET framework (including libraries and the common language runtime - CLR) for the server to run. A free, redistributable version of the .NET framework is available for download on http://www.microsoft.com/downloads/details.aspx?FamilyId=262D25E3-F589-4842-8157-034D1E7CF3A3&displaylang=en.