IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP5200

This paper appears in *Managing Modern Organizations Through Information Technology*, Proceedings of the 2005 Information Resources Management Association International Conference, edited by Mehdi Khosrow-Pour.

A .NET Remoting Medical Record Management System

Farid Hallouche and Matt Wendling

Dept of Compt. Sci. & Info. Systems, Saginaw Valley State University, 7400 Bay Rd., University Ctr, MI 48710, USA, hallouch@svsu.edu

ABSTRACT

The work presented here demonstrates that .Net Remoting is a simple yet powerful tool that enables disparate applications on different platforms to communicate transparently in a distributed architecture. This has the advantage of eliminating time-consuming and often-complicated development required for distributed applications found in earlier protocols such as DCOM and COM+. We present the design and implementation of a client/server application based on the .NET Remoting architecture. Using C#, a database-driven solution, the Medical Record Management System (MRMS), is developed that allows patients' medical records to be entered, updated, stored, and accessed automatically, and in a fine distributed environment.

INTRODUCTION TO .NET REMOTING

The .Net Remoting technology presents a myriad of very useful features and provides an infrastructure that includes various hosting options, message formatters, transport channels, lifetime management, and activation policies. Unlike older enablers, .Net is not a proprietary binary protocol. This makes it possible for an application to work across any platform. .Net Remoting is adaptable to different transport protocol formats, as well as different communication protocols. It can make numerous calls from the client and also supports callbacks. Since .Net Remoting is a homogeneous environment, the client must be built using a framework that is supported by .Net Remoting. The channels are used to transport messages to and from the remote object. The most commonly used channels are the HTTP Channel, the TCP Channel, and the SMTP Channel. The messages are encoded and then decoded before being transported by the channel. The .Net Remoting provides two message encoding schemes; binary encoding and SOAP encoding. While the binary encoding is slightly faster, the SOAP encoding is more suitable to electronic communication over the internet. A server object is created as a listener and accepts requests from remote objects. The server must be bound to an unused port. A request from a client-activated object will activate the server-side object. The "new" operator requests a message to be sent to the remote application. Objects are passed from one application to another by method calls using parameters, return values, or by reference from a field access. Once the remotable and server objects have been instantiated, the client will be able to connect to the server.

.NET REMOTING VS. DCOM

Distributed Computing Environment (DCE) / Remote Procedure Calls (RPC) has been the cornerstone for many higher level protocols such as DCOM and COM+ used in recent years for distributed applications. Applications used primarily for file and printer sharing also incorporated the DCE/RPC protocol. These applications include MS SQL Server, MS Exchange Server, and Network File System (NFS). The Distributed Component Object Model (DCOM) is an extension of the Component Object Model (COM) and uses a pinging process to manage the objects lifetime. As long as the client is sending messages via these objects, the server will maintain the object otherwise it will be destroyed. DCOM requires a direct TCP connection since it is based on the DCE/RPC protocol and makes the use of HTTP proxies impossible. COM+ also formally known as Microsoft Transaction Server (MTS) provided a Remoting platform as well as security and deployment services. However, COM+ does not support the automatic marshalling of objects (passing objects between applications by value); instead data structures are passed using ADO recordsets or by other means of serialization. Thus COM+ can be complicated and time consuming for the developer to configure and implement.

Before the advent of .NET Remoting, Microsoft used primarily DCOM for distributed applications. .NET Remoting overcome many of the DCOM difficulties. Because DCOM depends on proprietary binary protocol, it presents problems with regards to inter-platform operability as well as connectivity through the internet. Also, .NET remoting is much easier to learn, deploy, maintain and extend than DCOM.

Table 1 summarizes some of the differences between .NET Remoting and DCOM.

DESCRIPTION OF MRMS

In the medical field, one major area of practice that could definitely benefit from the application of appropriate information technology systems would be that of patient records. Many medical offices still keep and access medical records on paper. This system that has been widespread for many years, could very well improve its overall efficiency and convenience of accessing medical records through the use of

Table 1. Comparison of .NET Remoting and DCOM

DCOM	.NET Remoting
* Requires a direct TCP connection. * Uses a pinging process to manage object's lifetime. * Works best when similar applications are within the same network. * Has serious connectivity problems when applications distribute through different network environments or the internet. * Offers little extensibility. * It is difficult to log DCOM messages * Hard to learn, deploy and maintain. * Difficult to change channel configuration, or control message formats.	* Uses HTTP and TCP default channels. * HTTPChannel uses HTTP protocol, and formats messages using SOAP. * TCPChannel uses TCP protocol, and uses binary format for messages. * Uses simple and efficient leasing approaches to manage object's lifetime. * Makes it easier to build .NET applications, or web services across different platforms. * Easier to learn, deploy, maintain, and provides an extensible framework. * Allows change of channel configuration and message formats. * Allows HTTPChannel and TCPChannel to be extended, or a new channel created. * Because of HTTP, it is firewall friendly.

computer systems. Indeed, by creating a centralized data server for storing all records electronically, not only would medical records be much easier and faster to access by medial doctors, but also by other authorized parties as well such as pharmacies, medical labs, etc.

Our work is a contribution to demonstrate the application of a fairly new distributed technology, .NET Remoting, to the medical record problem. The system that is being implemented, the Medical Record Management System (MRMS) (Figure 1), is comprised of both client and server side applications. In general, the client software logs into the system and sends requests to the server software to view and/or modify a certain patient's health record. The server acknowledges these requests, makes sure they are valid, and performs the requested data operations. The server uses a database to store all of the electronic records, and also requires an application to allow for client offices to connect to it as needed to access these records.

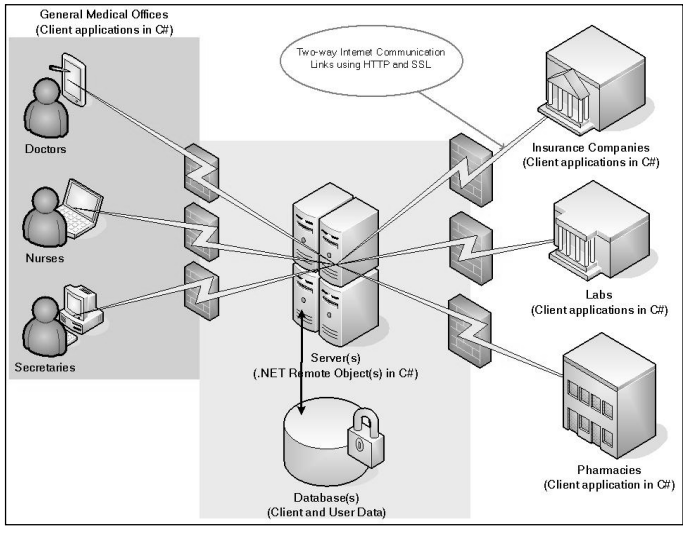
The MRMS is composed of a centralized server which stores client login information as well as patient records. Multiple clients currently have simultaneous, two-way communication with the server, which has been tested over a LAN connection. Clients can also connect over the Internet and through secure firewalls. Figure 1 shows a schematic block diagram of the MRMS.

The client program is developed in Visual Studio .NET using C#. It runs on a Windows platform using the .NET Framework and Remoting. The program runs in a standard window and requires the user to sign in with a personalized username and password. Each user will have a customized level of access to certain patient records depending on their position in the medical field. Whatever this access level may be, the interface of the client application will provide easy access to all functions that any particular user is allowed to perform.

The server side application is also developed using C#, for a Windows 2003 environment. The form of this application is a remote object hosted in IIS. Currently, clients can make remote calls to this object to process login information as well as viewing, modifying, and adding to user, office, and general patient information. This information is currently being stored in a database on the server using Microsoft Desktop Engine (MSDE) or SQL server. ADO.NET is used on the server-end to access information stored in databases. In order to secure communications with client programs, SSL is implemented. Figure 1 shows a simple general flow of communication within the MRMS.

Currently, the MRMS's server contains only a single database and two remote objects. Both will be discussed here in further detail, including information on how to configure both the client and server applications.

Figure 1. Block Diagram of MRMS



The Server Database

The database currently consists of two general categories of tables: The user, login, and office tables are used mainly to store information pertaining to the system users and their facilities. The patient, conditions, prescriptions, primary visit, and secondary visit tables are used to store the actual patient record information. These sets of tables could easily be divided into several different databases to better distribute server tasks.

The Remote Objects

The other major role of the server is to host the remote objects. This is how the clients will connect to the server and access the information stored in the databases. The two remote objects were programmed in C#, compiled into .dlls, and hosted as a web service using Internet Information Service (IIS) and the SOAP formatter. HTTP is used as the network protocol to avoid firewall issues and so SSL could be easily implemented. The RecordManager remote object handles most of the patient record queries, and the RemoteLogin object deals mainly with system information and user authentication. Again, it is very easy to distribute the server application by simply using multiple remote objects.

Configuration of the Server

The specific hosting environment used for the server on this project is Windows Server 2003. The operating system, along with IIS and SQL Server 2000, was installed on a personal notebook computer to simulate the server. After the remote objects were compiled, the following external XML configuration file was created to define the remote object as a web service.

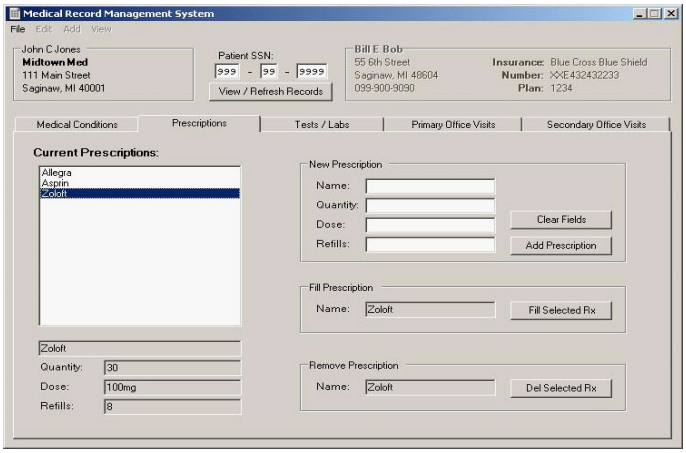
```
<configuration>
    <system.runtime.remoting>
      <application>
        <service>
          <wellknown mode="SingleCall"
type="RemoteLogin, RemLog" objectUri="RemLog.soap" />
          <wellknown mode="SingleCall"
type="RecordManager, RecMan"
objectUri="RecMan.soap" />
        </service>
      </application>
    </system.runtime.remoting>
</configuration>
```

This file was then placed in a directory along with another folder called bin. Inside of bin, RecMan.dll and RemLog.dll were placed, along with the .dll of the class library created for the system. A virtual directory was then created in IIS Manager which points to the directory containing all of this Remote Object information. At this point, the server is ready and waiting for clients to make calls to the hosted objects.

The Client Interface

The client in the MRMS (Figure 2) is mainly a graphical interface that allows users to log into the system to view and manipulate patient records. The level of control that each user has depends on his or her position in the medical field. For instance, a hospital can quickly and easily view patient conditions and medical history in emergencies. However, a hospital should not be allowed to add offices, users, and patients into the system as these are tasks of a system administrator. Furthermore, pharmacies should only be allowed to view and fill prescriptions, and labs should only be allowed to view and perform tests

Figure 2. Main Client GUI for the MRMS



as prescribed by a doctor. Obviously, each level of access must be carefully and thoroughly thought out and implemented.

Configuration of the Client

The client must also have its connection to the server properly configured. Again, a simple XML configuration file does most of the work. An external configuration file also allows the server's IP address to be manually changed relatively easily, as well as the used protocol, port, etc. The configuration file below is used when the client and server are being run on the same machine.

```
<configuration>
  <system.runtime.remoting>
    <application name = "ClientGUI">
      <client>
        <wellknown type="IRemoteLogin, IRemLog"
url="https://127.0.0.1/RemObjs/RemLog.soap" />
        <wellknown type="IRecordManager, IRecMan"
url="https://127.0.0.1/RemObjs/RecMan.soap" />
      </client>
      <channels>
        <channel ref="http" port="0" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

This file must be located in the same directory as the client executable, and the IP address must be set to the address of the server, instead of the loopback address 127.0.0.1. The compiled .dll for the created class library, as well as the .dlls for the remote objects' interfaces must also reside in this directory. The second part of the client configuration process can be seen in the following code snippets (Figures 3, 4).

Once this configuration is complete, the remoteLogin and recordManager proxy interface objects can be used just as a typical object, only the calls will be fulfilled by the object on the server. Figures 5a and 5b give an example of this communication. PatientCls is defined in the class library of the system, and is responsible for holding all information on a single patient. A SSN is stored in the PatientCls and then it is passed by

Figure 3. GetObject Function Assists in Configuring a Proxy Object for the Client

```
private bool Initialized;
private IDictionary WellKnownTypes;
private static Object GetObject(Type type) {
  if (!Initialized){
      Initialized = true;
      WellKnownTypes = new Hashtable();
      foreach (WellKnownClientTypeEntry entr in
          RemotingConfiguration.GetRegisteredWellKno
          wnClientTypes())
              WellKnownTypes.Add
              (entr.ObjectType,entr); }
WellKnownClientTypeEntry entr =
(WellKnownClientTypeEntry) WellKnownTypes[type];
  return
Activator.GetObject(entr.ObjectType,entr.ObjectUrl); }
```

Figure 4. Example Code Using GetObject to Connect the Proxy Interfaces to the Remote Objects

```
RemotingConfiguration.Configure("MedicalRecords.exe.
config");
IRemoteLogin remoteLogin =
(IRemoteLogin)GetObject(typeof(IRemoteLogin));
IRecordManager recordManager =
(IRecordManager)GetObject(typeof(IRecordManager));
```

Figure 5a. Example Client Code Calling fillPatient to Retrieve Patient Record Information

```
PatientCls currentPatient = new PatientCls( );
currentPatient.setSSN(ssn);
  recordManager.fillPatient(ref currentPatient);
```

reference to the remote object. The remote object then queries the database for all information on that patient and returns true if the patient was successfully populated, or false otherwise. Because the PatientCls was sent by references, the changes made on the server will be reflected in the local copy.

Secure Communication

Due to the confidential nature of patient records, a system such as our proposed MRMS would obviously need to implement an extensive amount of security. Possibly the most important and relevant area of security for our study deals with the communication channels between client applications and the central server. Fortunately, like many other aspects of .NET Remoting, implementing such security measures is relatively simple. When remote objects are hosted in IIS, as is the case with our MRMS, secure communication can be guaranteed by simply using SSL certificates

Secure Socket Layers (SSL) is a protocol that provides a secure channel (a secure TCP connection) between two internetwork hosts. It has quickly become the standard for authenticating and encrypting communications between Websites and Client Web Browsers. Almost every web browser and web server worldwide supports web transactions using SSL. Besides its famous use for securing web traffic, SSL is also used to secure file transfer (FTP), email transmission (SMTP), directory access (LDAP), etc. To ensure its reliable operation, SSL runs on top of TCP. In HTTP traffic, the client opens a TCP connection, on top of which an SSL channel is secured. Then, the HTTP request is sent over the SSL path. The server also responds through the SSL medium. Upon connecting to a secure server, a client receives a digital certificate authenticating

Figure 5b. Example fillPatient Function from the recordManager Remote Object Class

```
        public string strConn =
            "Provider=SQLOLEDB;DataSource=" +
            "NOTEBOOK2k3\\VSDOTNET;" + "Initial
            Catalog=medical;" + "User ID=sa";
    public bool fillPatient(ref PatientCls patient){
OleDbConnection cn = new
        OleDbConnection(strConn);
        cn.Open();
            string strSQL = "SELECT fname, lname, mi,
                address, city, state, zip, phone, doctor,
                iname, " + "inumber, iplan FROM
                patients WHERE ssn = '" +
                patient.getSSN() + "'";
        OleDbCommand cmd = new
            OleDbCommand(strSQL,cn);
        OleDbDataReader rdr = cmd.ExecuteReader( );
            if(rdr.Read()){
              patient.setName( rdr["fname"].ToString( ),
                rdr["lname"].ToString( ),
                rdr["mi"].ToString( ) );
              patient.setAddress( rdr["address"].ToString( ),
                rdr["city"].ToString( ),
                rdr["state"].ToString( ),
                rdr["zip"].ToString( ) );
              patient.setPhone( rdr["phone"].ToString( ) );
              patient.setDoctor( rdr["doctor"].ToString( ) );
              patient.setInsurance( rdr["iname"].ToString( ),
                rdr["inumber"].ToString( ),
                rdr["iplan"].ToString( ) ); }
    else {
        cn.Close();
        return false; }
        cn.Close();
        return true; }
```

the site. Subsequently, the client generates a unique session key that will encrypt the communications with the site. The client then encrypts the session key with the site's public key so that only the site knows how to read any information passed to it.

There is no built-in security in .NET Remoting. SSL works the same way when applied to .NET Remoting. For our MRMS, however, some extra work was needed in addition to setting up the server. Because SSL certificates must be purchased from a Certificate Authority (CA), we instead chose to work with Open SSL and create and sign a certificate ourselves. Because clients will not recognize this self-signed certificate as being from a trusted CA, the SSL Certificate Policy must be changed to allow the use of the certificate. Furthermore, while testing the system, we could not guarantee that the IP address of the server would always match the certificate. Figure 6 shows a class that changes the certificate policy to account for these problems. Also, the security features of a hosting Internet Information Services (IIS) can be used.

CONCLUSION

The combination of the expanding Internet and the onset of object-oriented programming have created the need for internetwork communication between software components and objects. Microsoft's .NET Framework and Remoting not only fulfill this need, but also do so in a manner that makes programming such applications relatively simple and highly customizable. Being a new technology, perhaps not many large-scale, real-world applications currently exist that use .NET Remoting. Our MRMS, which allows patient records to be accessed from any location, would be an ideal example use for such a powerful and highly promising technology as .NET Remoting.

Figure 6. Changing Certificate Policies in C#

```
public class MyPolicy : ICertificatePolicy
{
    public bool CheckValidationResult(ServicePoint
    srvPoint, X509Certificate certificate,
              WebRequest request, int certificateProblem)
    {
      string untrustedRoot = "800B0109";
      string cnNoMatch = "800B010F";

      int untrustedRootVal =
      Convert.ToInt32(untrustedRoot,16);
      int cnNoMatchVal = Convert.ToInt32(cnNoMatch,16);

      if (certificateProblem == 0 || certificateProblem ==
      untrustedRootVal || certificateProblem ==
      cnNoMatchVal)
          return true;
      else
          return false;
    }
}
```

Several key aspects of the .NET Remoting technology address issues of concern for such a medical system. For example, when dealing with medical records, confidentiality is a must; therefore, the additional tier of security added by .NET Remoting would prove to be quite useful. While most medical records could still be stored in a conventional database, the client application would first have to access the remote object on the server, which in turn would make any needed queries to the database, and then return the results to the client. By adding this extra step to the system, any potential hackers would have to get through multiple levels of security to access any confidential information. Also, by choosing to use remote objects to process client requests and access the database, any changes or additions to the system are less likely to have an impact on the client applications. Furthermore, the transmission of information between the client and server must also be kept secure. .NET Remoting allows for both custom security measures, as well as means to easily implement existing encryption technologies, such as SSL, into the system.

The legacy technologies such as COM, COM+, and DCOM were much more cumbersome when working with distributed applications. Our work is a feasibility study that demonstrates the benefits of such technologies by testing them through a real-life solution. More research work is envisaged related to remoting channels, database access, user access rights, developing solutions for heterogeneous network systems, and XML services.

REFERENCES

[1] Asaduzzaman Ahm "Forget the DCOM Pain and Use Remoting or Web Services" 29 Sep 2003. [Internet] http://www.devarticles.com/c/a/Web-Services

[2] Forslund, David and David Kilman "The Virtual Patient Record: A Key to Distributed Healthcare and Telemedicine" 29 Feb 1996. [Internet] http://openemed.net/background/TeleMed/Papers/virtual.html

[3] Longman, Eric "Self-Signed IIS SSL Certificates using OpenSSL" 02 June 2003. [Internet] http://eal.us/blog/_archives/2003/6/2/25109.html

[4] Rescola, Eric "SSL and TLS: Designing and Building Secure Ssystems", Addison Wesley, 2001, 0-201-615983.