TICKETING DISTRIBUTED SYSTEM USING .NET REMOTING AND WEB SERVICE TECHNOLOGY TO PROVIDE BASIC FAULT TOLERANCE ON THE MIDDLE SERVER

Ng Melissa Angga, M.M.Comp.
Universitas Surabaya
mels.ng@gmail.com

ABSTRAK

Kata kunci: .NET Remoting, Web Service, Client, Server, Fault Tolerance

1. Introduction
Nowadays distributed computing has been broadly implemented as a better alternative system than the centralized traditional system where the client only communicate with a single main computer like a mainframe. Some technologies provided to support distributed computing are .Net remoting for implementation on the .Net platform and the web service technology which could be established and accessed via multi platform.

It has been said widely that a good distributed computing system should provide an alternative solution to respond to a client if something has gone wrong with the server. This research is trying to provide an alternative solution by implementing the basic fault tolerance in the shape of the provision of a second alternative communication channel. A distributed system for ticketing process would be constructed as a start. The system consists of one main server for the performance developer server, one middle server for the agent of ticketing process and a number of clients from several devices (PC, web, and handheld device). Communication on this system would be established by means of .Net remoting and web service to provide a basic fault tolerance, so that the system keeps running well and to hide the problem arisly in times when something happened at the middle server.

2. Distributed System and Fault Tolerance
With the growth of business around the world, enterprises are challenged to broaden their connectivity. Enterprises need to reach their customers as well as their partners without the limitation of time and space. Regardless of when or where, the communication should be able to establish. This would be the right time where the network technology fits in as a result.

The distributed system is defined as a collection of autonomous computers which connect to each other through a network and are supported with software that has been designed to provide an integrated computing facility[1]. It was built with the need of resource sharing, as Colouris (2001) stated. Using distributed system, enterprise could reduce the cost by distributing process through inexpensive systems while reducing server workload. One characteristic of a fine distributed system is to put on how the system handles a failure. A good distributed system has to find a way out if one part of the system has failed to perform correctly. This is a basic fault tolerance in which the situation could be rolled back to the previous condition before the failure began and the system has to try another alternative while avoiding the failure that has been discovered previously.

3. Net Remoting
The idea of .NET remoting is to invoke a remote object without being restricted by the application domains, processes, or machine boundaries[7]. The term used for transporting a remote object from one to another is marshalling[8]. In order to be marshalled, a class definition for objects which has prepared to roam across the network should inherit the System.MarshalByRefObject class.
To apply .NET remoting procedure, first of all the systems need to register at the channel used which can be the tcp or http channel, complete with the port number. The next thing to do is to register the object class available to be invoked remotely. While registering such object, the system has to decide whether the object type would be a single call or singleton. A single call object is created for each call, whereas a singleton object is created once for all[7].

At the time when a client wishes to call a method of a remote object, a proxy created previously using the object definition is used to invoke the methods[7]. This situation was hidden from the client, and the client has a possibility to call the object methods as if it is located in the same environment.

4. Web Service
Web service technology enable people to create a method somewhere in this world using some kinds of platforms, while other people from another place on another platform could build an application using those methods created previously. There is no need to implement those methods again and simply reuse somebody else’s method. Since the web services use the standard protocols and ride on the web, they have independent characteristics that are platform independent and language independent. Web service creator can place their services regardless of what language they are using or in what platform they are built on. At the same way, the web service client could always access the service without any regard to the language and platform they use.

Technically, web services are standardized methods for web-based application interoperability using the open standards such as Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery and Integration (UDDI) specification[4]. XML is used as the “uniform data representation and exchange” of web services which is the format used in SOAP as the “standard communication protocol” (Figure 1)[3]. WSDL acts as a “standard meta language” used as a description of the service provided. UDDI takes part as the directory services tool to register and locate a web service that has been deployed.

![Figure 1. Web Services Architecture](image)

4.1 Web Service Essential
The discussion about web services usually concerns the four basic issues i.e. the Service Description, Service Implementation, Service Publishing Discovery and Binding and Service Invocation and Execution[5]. The term of service description refers to the ability of web services to describe themselves in standard ways in order to support the reusability concept[3]. This role has been coped by Web Service Definition Language (WSDL) whose responsibility is to specify web services as a group of “message-enabled endpoints” using XML grammar. WSDL could be used to describe the interface of a service which is usually needed by developers while they build applications bound to a web service (Gardner, 2001).

Referring to the service implementation of web service technology, there is a need to configure the data and operations in XML documents that meet the terms of SOAP specification[5]. After the web service component is implemented, the web service users or clients could merely send an XML document to web service as a request and then accept the clients’ response from the component as an XML document.

Web service needs to publish itself to help other parties in finding it and expose information for them about how to access and interact with it[5]. The information about how to connect and interact with a particular web service is called the binding information and is located on the web service registry. An interested client would first look up the registry to find the binding information which can be used to invoke the desired service[6].

178
The invocation and execution of a web service is based on Simple Object Access Protocol (SOAP), which is basically an XML-based protocol for exchanging structured data and information type through the web [5]. Web service addresses would act as a SOAP listener and would send information to the concerned parties after receiving a request. The listener would authorize the SOAP message based on its XML schemas described in the WSDL before unmarshalling it. After it invokes the business logic through the message dispatcher, the reply would be sent back to the caller after it is wrapped in SOAP.

4.2 Web Service and .NET
Creating web services using ASP.NET is quite simple as can be seen from the example. For the first time developer who has never dealt with web services in .NET framework, Visual studio even generates a sample code to show the format used in building the web service.

```csharp
Imports System.Web.Services

<Service Contract="http://tempuri.org/MyService/Service1">
    <Service Operation="http://tempuri.org/MyService/Service1">
        <OperationContract>
            <WebMethod()> ' WEB SERVICE EXAMPLE
            ' The HelloWorld() example service returns the string Hello World.
            ' To build, uncomment the following lines then save and build the project.
            ' To test this web service, ensure that the .asmx file is the start page
            ' and press F5.
            ' 'WebMethod()'
            'Public Function HelloWorld() As String
            'Return "Hello World"
            End Function
        </OperationContract>
    </Service>
</Service>
```

Figure 2. Default Web Service Sample Code Generated By Visual Studio

The next tasks, such as creating the WSDL and discovering files would automatically be handled by .NET framework. This approach makes the application more convenient for developers since they do not have to write such files.

5. System Design and Implementation
The ticket selling system is a good example of a distributed system. On one side, a main server acts on behalf of the producers and publishes its performance to some middle servers as agents which are responsible to sell tickets. The agent server is the one who is really connected with the clients. The agent server acts as the middle server and obtains information about events produced by the main server.

The client is then able to access those information. If the client is interested in one particular event and decides to book some tickets, the client needs to send some details to the agent server. The agent server would then pass the details provided to the main server, which is used to decide the validity of the booking process. After that, the main server would send the response to the agent server who accordingly would pass the response to the client. In this case the agent server (the middle server) would take the role as a server and a client at the same time. To manage this issue, the system use both http and tcp channels, one for the communication with the main server and the other for the communication with the client side.

5.1 Main Server
UML class diagram for the class used in the main server is as shown in Figure 2. The main server was build from several classes which are separated as the business logic, the data management layer, the network layer, and the application itself. The business logic was presented through the DatabaseOperation class which handles all the operation on the data storing on the main server. The class DatabaseClass which provides all operations for retrieving and manipulating the database was a presentation of a data management layer. ITicket and Ticket would be the object and the presentation which are invoked by remote systems. The class consists of two methods for obtaining the newest event and performing the booking process. Lastly the communication layer was represented through the Communication class.
Figure 3. Main Server UML

Figure 4. Middle Server UML
The implementation of the remoting object connection has been done by registering the HTTP channel and the port number used to communicate (ChannelServices.RegisterChannel method). It is followed by the command to register the TicketData class as the singleton marshaled object (RemotingConfiguration.RegisterWellKnownServiceType method). At this stage, the fault tolerance is also implemented in the .Net remoting process by providing two different port numbers for the HTTP channel. If one port number failed to perform then the system would try to connect using another port number.

5.2 Ticketing Server

UML class diagram for the middle server is shown in Figure 4. The ITicket interface has to be included in the references library since it is the form of the object on the main server being invoked. The IDataAccess and DataAccess class would be the object invoked remotely from the client side. It consists of methods for retrieving the event tables, the booking process and the customer login process.

Similar to the main server, there is a business logic layer performed by the AgentData and the DataHandling class. The communication layer is performed using the AcceptInvoke and InvokeServer class. And the data management layer represented with the DatabaseClass class. To invoke the Ticket object from the middle server, the system needs to call the Activator.GetObject method in the communication layer. The fault tolerance in this part is included in the part where the activator tries to build the connection first via port number 8118. If it fails, the system simply moves the number to port number 8338. The implementation of the marshaled object connection with minor fault tolerance on the middle server side is established using the same method as the main server.

The web service implementation for the middle server is almost the same as creating a class definition. In this case, it is the DataAccess class. To switch it to a web service, the class should inherit System.Web.Service.WebService class. The class then completes the process by simply adding the web service header (<System.Web.Services.WebService (Namespace:="http://tempuri.org/DataAccess/DataHandling")>) on the line before the class header and a web method header (<WebMethod()> _ ) before each method header.

5.3 Multi-Client

The UML class diagram for the multi-client is almost similar the above diagram. Therefore it is presented only using one example.

![Diagram of Web Client UML](image)
The implementation of the fault tolerance on the client side was provided by invoking the remoting object first. If it fails or it throws any exception, it then rolls back and uses the implementation of the web service. Before the web service is available on the client side, it has to be added as a web reference first.

6. Evaluation
The evaluation for this research is established first by observing the performance of the overall system, when the communication is provided via .Net remoting. The evaluation is also performed in order to check the fault tolerance issues when the system needs to communicate via the web service at the condition when the attempt to connect to the middle server failed.

6.1 Using .Net Remoting

From the windows application, the user can find performances which are available from the list and then, if needed, the user can see the detail of performances chosen. After choosing one performance, the user could book the preference performance by clicking on the “Book Now!” button which would lead to the next window. After the user fills in all the details needed to book the tickets, they need to press the “OK” button. The application will try to invoke the object on the main server to do the booking method. If the booking process is successful, the user will be informed and get the total price and the transaction number information required. This is shown in Figure 7.

![Figure 6. Select Performance Window](image)

![Figure 7. Booking Window](image)
The operation performed in a web application is similar with the one used in a windows application. But the user interface and the platform are different. Figure 8 shows the test on the booking process of the web application. The same operations are also performed in smart devices. Figure 9 shows a screen shot of the process for choosing a performance and booking operations.

6.2 Fault Tolerance Evaluation

The evaluation is usually established at the time when the middle server is on. Now the evaluation will be arranged in the situation where the middle server is down. In that case the system should roll back and use another way i.e. the web service. The result shows that nothing seems to change at the client side, while in reality there is a problem on the middle server side. The problem is hidden from the client sight. The same result has happened on the web and handheld applications as well.

7. Conclusion

While performing two evaluation conditions by letting the middle server on and shutting down the middle server, the result has shown that the system can run well on both conditions. Even when the middle server has failed, the client application still performed as if the middle server still works fine. From the user point of view, there is nothing different between two circumstances. It could be concluded that a basic fault tolerance could be achieved using both .NET remoting and web service technology at the same time. This capability could be reached since the web service is usually placed on the web server side. Therefore if the server is down, the communication could still be established by relying on the web server.
Bibliography


