Microsoft .NET Distributed Technology

Sukasom C.
School of Informatics
University of Edinburgh

I. Introduction

The concept of Service Oriented Architecture (SOA) is getting popular today. It took 40 years to achieve the point where we can write software component to be more modular enough to represent a service [8] [18]. In SOA architecture, each department provides services. The whole SOA infrastructure is easy to maintain, because each service is maintained independently from various departments. It is easy to remove or add a service to an existing infrastructure without much trouble to other existing services. Distributed technology is needed to connect services in an infrastructure together to perform a business task.

Distributed technology is used to enable an application to extend its process to more than one computer, and its goal is to improve performance and scalability [9]. There are many distributed technologies running on various platform including Microsoft and Java. This paper focusses on aspect of programmability of the current generation of distributed technology which comprises .NET Web Service, .NET Remoting and Microsoft Message Queuing (MSMQ) from Microsoft based on .NET Framework. Different technologies and different platforms have different programming models. Programmability aspect is important to todays business. Something which is easy to build or maintain helps keeping the cost of development and maintenance down. It also can help the company to move forward faster in terms of information technology.

Sun Microsystem also made a lot of distributed technology specification including Java Web Service, Enterprise Java Bean (EJB), and Java Message Service (JMS). There are a lot of vendors implementing these specification such as Oracle, IBM, BEA, or Sun Microsystem itself. Vendors typically sell their implementations in the form of application servers. These implementations provide APIs for implementing Java distributed technologies to run on application servers. Not all vendors adheres to the specifications, so it is possible that a distributed application may not work 100% on different application servers from different vendors. Finding programmers with particular knowledge in specific application servers or environment editors is difficult, because there are so many tools and application servers. For Microsoft, knowledge can be consolidated through one tool, Microsoft Visual Studio (Visual Studio), and one application server, Microsoft Internet Information Services (IIS).

An overview architecture of each technology is given to aid the reader's understanding. The paper also compares each .NET distributed technology with a relevant Java distributed technology. The paper also gives a quick overview of the next generation .NET distributed technology which is Window Communication Foundation (WCF).

Section 2 discusses .NET Web Service distributed technology. Section 3 discusses .NET Remoting distributed technology. Section 4 discusses Microsoft Message Queuing (MSMQ) distributed technology. Section 5 gives a quick overview of the next generation distributed technology from Microsoft, Windows Communication Framework (WCF). Section 6 concludes the paper.

II. .NET Web Service

Web Services is a technology that enables an application to call services on another computer. Web Services usually uses the HTTP protocol as a transport mechanism. A web service server and a client can reside across the Internet. Requests and responses to web services do not usually get blocked by firewalls, because firewalls do not usually block HTTP ports.

A. An Architecture Overview

There are two main parties involved in the Web Services architecture. These are a web service server who provides a service and a client who consumes a service. Fig. 1 illustrated a scenario of using a web service. In the scenario, there is the XML web service in the bottom left and the ASP.NET web application in the middle. The ASP.NET web application, a consumer, calls the XML web service, a provider. It is simply requests and responses between web service providers and consumers.

A web service makes use of the SOAP (Simple Object Access Protocol) protocol to transport a message. Data in a SOAP message is explainable in itself, but the size of a SOAP message is larger than the size of a binary file for the same amount of data. Fig. 2 illustrates main components in a SOAP message. A SOAP Message can be considered as an envelope containing a header and a body [16] [3] [6]. The header section is usually stores metadata or credential data such as a certificate. The body section stores data. Web services uses the HTTP to transport a SOAP message. Other protocols such as the SMTP can be configured to transport a SOAP message as well.

A web service may publish its accessing information in a Web Service Definition Language (WSDL) file to a Universal Description Discovery and Integration (UDDI) server. Developers can query the UDDI server for a WSDL file of the desired web service. In practical, developers usually obtain

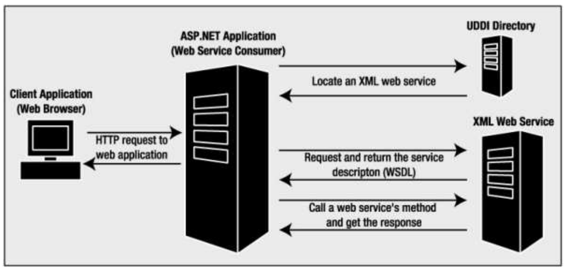


Fig. 1. Full life cycle of web service. Source : MacDonald, Matthew, and Szpuszta, Mario: Pro ASP.NET 2.0 in C♯ 2005, Apress, 2005. 7 November 2007.

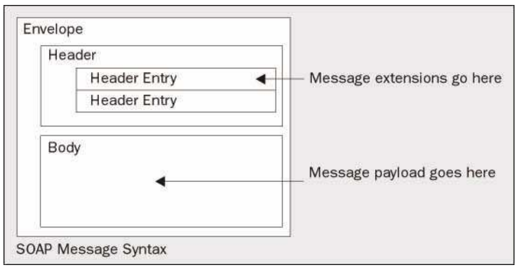


Fig. 2. SOAP components. Source : Cauldwell, Patrick, Chawla, Rajesh, Chopra, Vivek, Damschen, Gary, Dix, Chris, Hong, Tony, Norton, Francis, Ogbuji, Uche, Olander, Glenn, Richman, Mark A., Saunders, Kristy, and Zaev, Zoran: Professional XML Web Services, Apress, 2004. 8 November 2007.

a link to a WSDL file from other sources such as Internet or other developers. A WSDL file is needed by a client to generate a proxy class.

B. Development

Developers can turn a regular method in a regular class into a web service by applying the *WebMethod* attribute

```
public class EmployeesService :
            System.Web.Services.WebService
{
    [WebMethod()]
    public int GetEmployeesCount()
    { ... }

    [WebMethod()]
    public DataSet GetEmployees()
    { ... }
}
```

Fig. 3. Turning a method to a web service by applying the *webmethod* attribute. Source : MacDonald, Matthew, and Szpuszta, Mario: Pro ASP.NET 2.0 in C♯ 2005, Apress, 2005.

```
 <%@ WebService Language="C#"
     Class="EmployeesService" %>
```

Fig. 4. Declaring web service by using *WebService* directive in the aspx page. Source : MacDonald, Matthew, and Szpuszta, Mario: Pro ASP.NET 2.0 in C♯ 2005, Apress, 2005.

```
...
// Create the proxy.
EmployeesService proxy
              = new EmployeesService();
// Call the web service
//  and get the results.
DataSet ds = proxy.GetEmployees();
...
```

Fig. 5. A part of an example consuming the web service. Source : MacDonald, Matthew, and Szpuszta, Mario: Pro ASP.NET 2.0 in C♯ 2005, Apress, 2005.

above a method. Fig. 3 illustrates an example of turning functions in a normal C♯ class into a web service by placing the *WebMethod* attribute above the functions. A web service can return a simple data type like *int* or complex data type like *DataSet*. A class hosting a web method can optionally inherit the class *System.Web.Service.WebService*. Inheriting from *System.Web.Service.WebService* gives a class an access to predeclared ASP.NET object variables [10].

Developers need to declare a web service in an ASP.NET application. Declaration of the web service is done in a .asmx file with the *WebService* attribute. Developers must supply corresponding class through the *Class* attribute as shown in the Fig. 4. Deployment can be easily done by deploying the ASP.NET web application into the Internet Information Services (IIS) server. The IIS server is a web application server capable of running ASP.NET web applications.

Developers can view a WSDL file of a web service by adding ”WSDL” to the URL of the web service. Developers can either use wsdl.exe or Visual Studio to generate a proxy class [10]. Visual Studio hides a generated proxy class, because it wants to reduce the complexity of codes. Fig. 5 illustrates the process of consuming the web service in the Fig. 3. The name of the proxy class on the client, *EmployeesService*, is the same as the name of the web service class on the server. With a generated proxy class, a client application can treat a remote web service as a local service. Treating a remote object as a local object reduces the code complexity. A web service can be developed quickly with Visual Studio [5].

C. Comparison with Java Web Service

The architecture of the Java Web Services is the same as .NET Web Services. A Java web service also uses the SOAP protocol for transporting a message and the WSDL for describing its service interface. There are many implementations of Java web services API from various vendors including Sun, IBM, Oracle and Apache. One of the popular implementations is Axis from Apache Software Foundation [17]. Developers can use existing an API tool like Axis from Apache to help building a Java web service. Developing a Java web service without tools is a time consuming task. Using a tool like JBuilder Enterprise or IBM Websphere can ease development processes.

III. .NET Remoting

.NET Remoting can be used to host services or remoting objects. A client application treats a remote object as a local object. .NET Web Service is built for interoperability between different platforms and across the Internet. The .NET Remoting is recommended when a client and a server are implemented with .NET Framework, and both of them are in the same network. .NET Remoting can be configured to use three transport mechanisms which are HTTP, TCP and IPC (Inter-Process Communication).

A. Architecture Overview

.NET Remoting consists of a server which hosts a remote object and a client application. Fig. 6 shows an overview of the various components in the .NET Remoting architecture. A proxy is a representation of a remote object on a client machine. Clients call proxies. Proxies send a request to a formatter. There are two default formatters: SOAP and binary formatters. The SOAP formatter formats the data sent into the SOAP format. The binary formatter formats the data into the binary format. Using the binary formatter could obtain a better performance than using the SOAP formatter [9]. After formatting a request, the request is sent through a transport channel. The transport channel delivers the request to a formatter on a server which deserializes the request. Then the server sends the request to a hosted remoting object.

There are two mechanisms of how the server marshalling remote objects to clients which are *marshall-by-reference* (MBR) and *marshall-by-value* (MBV) [11].

- MBR : A remote object is created on a server. The server passes a reference to a client. The client accesses the remote object via a proxy. The proxy serializes a request from the client. Then the server deserializes the request and passes to an appropriate remote object [11]. MBR can be implemented by marking a class with the attribute *serializable* [15].
- MBV : A local copy of a remote object is passed to a client. The remote object may not have an access to resources on a server anymore when it is on the client. MBV can be implemented by inheriting a remote class from the *System.MarshalByRefObject* class. Because the entire remote object is serialized to the client, it exposes the internal detail of the remote object. MBV should be used in the case of improving performance [11], [15].

1) Remote Object Types: There are three types of .NET Remoting objects which are Single Call Objects, Singleton Objects and Client-Activate Objects [9] [11].

- Single Call Objects : A remote object does not keep state between multiple calls from clients. It is easy to scale simply by adding more remote objects to serve calls from clients.
- Singleton Objects : When there is only one remote object created on a server. This remote object is used to serve all client calls. It is used in the case when creating a remote object is more expensive than maintaining a remote object.
- Client-Activate Objects : A remote object is activated when a client needs.

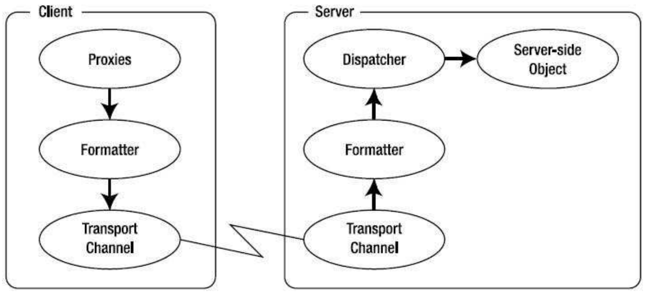


Fig. 6. .NET Remoting Architecture. Source : Rammer, Ingo, and Szpuszta, Mario, "The .NET Remoting architecture (simplified)." Advanced .NET Remoting, Second Edition, Apress, 2005. 6 November 2007.

2) Transport Channel: HTTP, TCP and IPC are the channels available for transporting data between a client and a server. Both SOAP and binary formatter can be configured to use with all channels.

- HTTP : The HTTP channel is suited to transport data across the Internet. The common formatter for the HTTP channel is the SOAP formatter.
- TCP : The TCP channel is usually blocked by firewalls. The common formatter for the TCP channel is the binary formatter.
- IPC : The IPC (Interprocess Communication) channel of Windows performs faster than the HTTP and the TCP if a client and a remote object are in the same machine [11]. The common formatter for the IPC channel is the binary formatter.

B. Development

Unlike developing a web service in Visual Studio, there is no ready template available in the Visual Studio for creating .NET remote objects. It is difficult to develop and deploy .NET Remoting compared with .NET Web Service [9].

First, developers simply need to create a class library project in Visual Studio for a remote object class. This class library project containing the remote object will be generated into a dll file. Then developers need to create another project for a server application. The server application is needed to host the remote object. In the server application Visual Studio project, developers need to add a reference to the remote object by adding its dll file into the project.

An example of a remote object is illustrated in Fig. 7. The dummy class *Patient* represents information about a patient. This class inherits from *System.MarshalByRefObject*, so this remote object is MBV type.

A configuration file is needed for a server application to host a remote object. Fig. 8 illustrates the configuration file. The *service* element represents a service offered. It specifies that the remote object is a *server-activated* object through the

```
namespace MSPress.Chapter4.Remoting.Server
{
  public class Patient :
                     MarshalByRefObject
  {
      public string PersonalInformation()
      {
         return "Patient 1: Demo Patient";
      }
  }
}
```

Fig. 7. An example of .NET remote object class. Source : Sarah Morgan, Shannon Horn, and Blomsma, Mark: MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development, Microsoft Press, 2007

wellknown element. The *wellknown* element also specifies a class of the remote object. The remote object type is *singleton*. The configuration file also specifies a channel which is TCP at port 9000 through the *channel* element.

There are four kinds of applications which can host remote objects: ASP.NET, Windows console, Windows GUI application, and Windows Service [12]. An ASP.NET application is run inside the IIS server. The IIS server provides web security framework. A server application hosted in the IIS will be started immediately when the IIS is started. A Windows console is the easiest server application type to create. However, it requires users to manually typing in a console to start the server. A Windows GUI requires users to manually start the server as well. A Windows service is a service which starts automatically when Windows starts. Hosting in the Windows service does not require users to start the server each time when Windows is started, but debugging with the Windows service could be more difficult than other types of server applications.

Fig. 9 demonstrates the client application consuming the remote object. First, the client application needs to create a channel to connect to the server by instantiating an instance of the *TcpClientChannel* class. Then the client application calls *RemotingConfiguration.RegisterWellKnownClientType* method to register the *Patient* remote object. After that, the client can simply use this remote object by using a new keyword, and the client application can treat the remote object as a local object.

Developers can deploy remote objects to clients by deploying an entire assembly or deploying only an interface. Deploying an entire assembly to clients is suitable for MBV. Deploying only an interface is suitable for MBR. Both deployment methods require developers to give an assembly file, a dll file, to clients, so client applications can reference remote objects as shown in the Fig. 10.

C. Comparison with Enterprise Java Bean (EJB) 3.0

We choose to compare .NET remoting with Enterprise Java Bean (EJB) 3.0, because they have some common. Both of

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.runtime.remoting>
    <application>
      <service>
        <wellknown mode="Singleton"
          type="...Patient, RemotePatient"
          objectUri="Patient.rem" />
      </service>
      <channels>
        <channel ref="tcp" port="9000" />
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

Fig. 8. An Example of .NET Remoting configuration file in a Server. Source : Sarah Morgan, Shannon Horn, and Blomsma, Mark: MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development, Microsoft Press, 2007

```
...
TcpClientChannel channel
     = new TcpClientChannel();

ChannelServices.RegisterChannel(
    channel,
    false);

RemotingConfiguration.
  RegisterWellKnownClientType(
     typeof(Patient),
    "tcp://localhost:9000/Patient.rem");

Patient newPatient = new Patient();
Console.WriteLine(
    newPatient.PersonalInformation());
...
```

Fig. 9. An example of using a .NET remote object in a client. Source : Sarah Morgan, Shannon Horn, and Blomsma, Mark: MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development, Microsoft Press, 2007

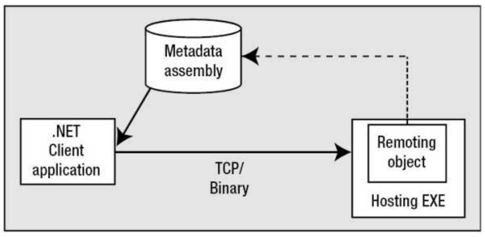


Fig. 10. Deployment overview of a .NET remote object assembly. Source : Titus, Tobin, Gilani, Syed Fahad, Gillespie, Mike, Hart, James, k. Mathew, Benny, Olsen, Andy, Curran, David, Pinnock, Jon, Pars, Robin, Ferracchiati, Fabio Claudio, Gopikrishna, Sandra, Redkar, Tejaswi, and Sivakumar, Srinivasa: Pro .NET 1.1 Remoting, Reflection, and Threading, Apress, 2005. 6 November 2007.

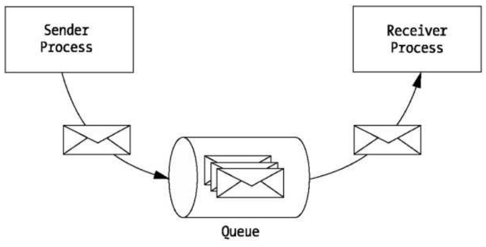


Fig. 11. Overview of MSMQ Architecture. Source : Arohi, Redkar, Carlos, Walzer, Scot, Boyd, Richard, Costall, Ken, Rabold, and Tejaswi, Redkar: Pro MSMQ: Microsoft Message Queue Programming, Apress, 2004. 7 November 2007.

them have the concept of remote objects. A client and a server must be in the same platform. EJB devides their remote objects into an entity bean, a session bean, a stateless bean and a message driven bean [14]. An entity bean represents an object which stores some related information together. A session bean and a stateless bean contain business methods. A message driven bean is used for receiving a message. EJB configuration can be done programmatically through annotation or through a configuration file. A client must first look up a remote object from a naming service first. A client also treats a remote object as a local object using the bean interface. There are many implementation of EJB containers from various vendors including IBM, Borland, BEA and Sun. A business method in a session bean and a stateless bean can be converted into a web service [7].

IV. Microsoft Message Queuing (MSMQ)

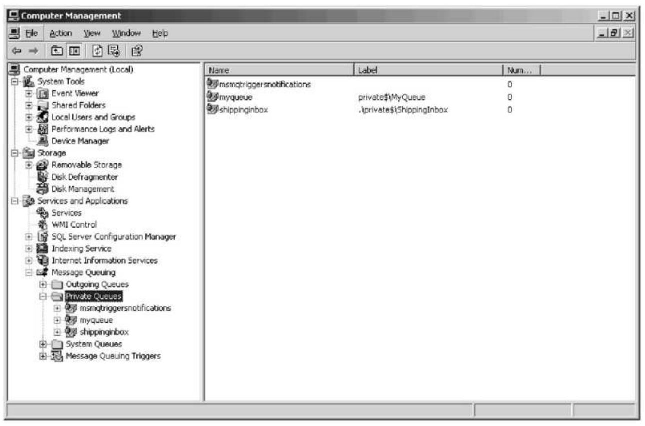
Developers can use Microsoft Message Queuing (MSMQ), Middleware, to send messages between processes on different computers. Every message is guaranteed delivery even networks are temporally offline. Message sent is stored on a queue on a remote server, and a receiver will read from the queue [2].

A. An Architecture Overview

Fig. 11 illustrates key players in a MSMQ scenario. The MSMQ architecture consists of a sender, a queue and a receiver. The sender sends a message to the receiver via the queue. There are 2 main kinds of queues which are a public queue and a private queue. A public queue is listed in an Active Directory. For a private queue, developers must know the address of the private queue. Using a private queue tends to be faster, because it does have to deal with an Active Directory [11]. There are two kinds of messaging: Express and Recoverable Messaging. Express messaging stores all messages on a random access memory (RAM), while Recoverable Messaging stores all messages on a physical hard disk. If a queue server or a client is down, Recoverable Message can continue sending messages after the queue server or the client is up.

B. Development

Developers can either program or configure to create a queue. Developers can use the Computer Management found



Name	Label	Num...
msmqtriggersnotifications		0
myqueue	private$\MyQueue	0
shippinginbox	.\private$\ShippingInbox	0

Fig. 12. Using Microsoft Management Console (MMC) to create or view a queue. Source : Sarah Morgan, Shannon Horn, and Blomsma, Mark: MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development, Microsoft Press, 2007. 7 November 2007.

```
...
string queueName
    = @".\private\ShippingInbox";

MessageQueue queue
    = MessageQueue.Create(queueName,true);

queue.Label = queueName;
...
```

Fig. 13. Part of the example of creating a queue in MSMQ. Source : Sarah Morgan, Shannon Horn, and Blomsma, Mark: MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development, Microsoft Press, 2007

```
...
ShippingOrder order
      = new ShippingOrder();

Message msg
      = new Message();

BinaryMessageFormatter formatter
      =  new BinaryMessageFormatter();

formatter.Write(msg, order);
...
```

Fig. 14. Part of the example of sending an object message to a queue in MSMQ. Source : Sarah Morgan, Shannon Horn, and Blomsma, Mark: MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development, Microsoft Press, 2007

```
...
Message msg
      = queue.Receive(tran);

BinaryMessageFormatter formatter
      = new BinaryMessageFormatter();

object body
      = formatter.Read(msg);
...
```

Fig. 15. Part of the example of sending a message to a queue in MSMQ. Source : Sarah Morgan, Shannon Horn, and Blomsma, Mark: MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development, Microsoft Press, 2007

in the Administrative Tool in Control Panel to view existing queues and create a new queue [1]. The Computer Management tool is shown in the Fig. 12. Developers can also use .NET supported languages like C♯ or Visual Basic to create a queue programmatically. Fig. 13 illustrates the process of creating a queue using C♯ programmatically. A queue can be create by using *MessageQueue.Create* static function of class *MessageQueue*. The first important parameter of the *MessageQueue.Create* is the name of the queue which is used to distinguish this queue from other queues. Fig. 14 illustrates a part of example of a sender application. Developers can write an object or a string into a message. In Fig. 14, an object of type *ShippingOrder* is written to the message. The *BinaryMessageFormatter* is needed to format the message into a binary form. Another possible formatter is *XMLMessageFormatter* [2]. Fig. 15 illustrates a part of example of a receiver application. The *Read* function of the *BinaryMessageFormatter* class is used for Converting a message into an object.

C. Comparison with Java Message Service (JMS)

JMS is an enterprise messaging where delivery is guaranteed. MSMQ is not Interoperability with Java Message Service (JMS) [7]. Both of them do the same work, sending a message. There are two main messaging architectures in JMS which are point-to-point and publish-subscribe [19]. The point-to-point architecture is used when an application wants to send a message to a specific application. The publish-subscribe architecture has a Topic which can be created by configuration on an application server. An application would send a message to the Topic, and then a subscribe application would a retrieve a message from the Topic. Same with Java Web Service, JMS is a specification. There are many API implementation from various vendors including from Sun and IBM. Developers can use those APIs to implement a JMS application easily.

V. A Quick Look at the Next Generation .NET Distributed Technology : Window Communication Foundation (WCF)

Developers can construct .NET Web Service, .NET Remoting and Microsoft Message Queuing (MSMQ) from .NET framework version 1.0, 2.0 and 3.0. There are only slight difference between versions. However, there is a lot of significant change in programming model in the new Windows Communication Framework (WCF). the WCF is a new framework for creating a distributed application. WCF is only available in .NET framework 3.0.

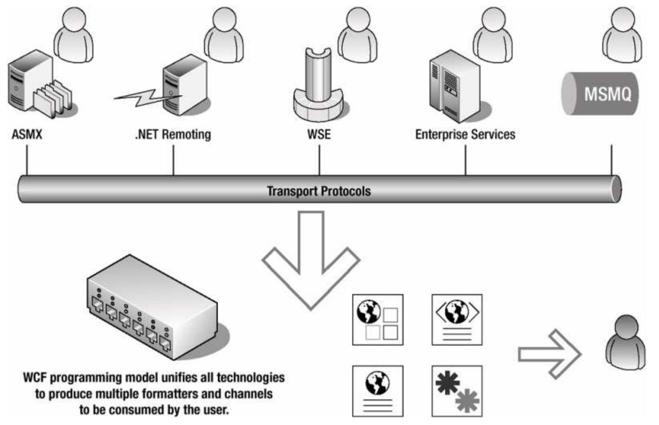

ASMX
.NET Remoting
WSE
Enterprise Services
MSMQ
Transport Protocols
WCF programming model unifies all technologies to produce multiple formatters and channels to be consumed by the user.

Fig. 16. Unification of existing .NET distributed technology. Source : Chris Peiris, Nishith Pathak: Pro WCF: Practical Microsoft SOA Implementation, Apress, 2007. 9 November 2007.

The WCF sits on top of the .NET framework providing distributed technology to an application. The WCF is designed to be extensible to support future communication standard [13].

The WCF unifies exiting distributed technologies into one programming model. Fig. 16 illustrates the WCF programming model. The WCF enables developers to use just one method of coding for various protocols and transports. For example, codes for sending a message for TCP/IP and MSMQ are the same in WCF. Previously, when programming MSMQ, developing and configuration MSMQ requires specific knowledge which can not be used for other .NET distributed technology. The benefit of unifying programming model helps shortening the learning curve of developers when using a new transport mechanism or protocol. WCF helps developers to build distributed system faster than before [13].

Unification of all programming models is possible with an endpoint. An endpoint is an interface of each service. The endpoint defines address, contract and binding. An address consists of an URL, a machine name, a path, and optionally a port number [4]. A service may have multiple endpoints where each endpoint may use a different transport mechanism. For example, a service may have 2 methods of accessing it. One is TCP and another is MSMQ. Therefore, TCP and MSMQ endpoints are needed to be declared in a configuration file as illustrated in Fig. 17 and Fig. 18. Technology specific coding is not needed for each method. The WCF makes it very convenient to add or change a transport mechanism.

VI. Conclusion

Microsoft offers three distinct distributed technologies which are .NET Web Service, .NET Remoting and MSMQ.

```
...
<endpoint
    address="net.tcp://.../Exchange"
    bindingsSectionName="NetTcpBinding"
    contract="IExchange" />
...
```

Fig. 17. Endpoint Configuration with TCP as a transport mechanism. Source : Chris Peiris, Nishith Pathak: Pro WCF: Practical Microsoft SOA Implementation, Apress, 2007

```
...
<endpoint
    address
      ="net.msmq://.../...SettleTrade"
    bindingsSectionName
      ="NetMsmqBinding"
    contract="IExchange" />
...
```

Fig. 18. Endpoint Configuration with MSMQ as a transport mechanism. Source : Chris Peiris, Nishith Pathak: Pro WCF: Practical Microsoft SOA Implementation, Apress, 2007

Each technology has a different programming model and a different architecture. Each technology has its own advantage and disadvantage. Given a requirement, developers must choose an appropriate technology to implement an application with in given time.

.NET web service mainly uses the HTTP protocol for sending messages. The HTTP protocol can bypass firewalls very well. .NET web service is suitable when a client application resides on the Internet or on a different platform. .NET Remoting is suitable when a client is in the same intranet and platform (.NET framework). .NET Remoting is supposed to be faster but less interoperability than Web Services. Messaging can be done with the Microsoft Message Queue (MSMQ). Messaging takes advantage of asynchronous communication. A client can send a message and forget about it, and the client can move on to do something else. MSMQ guarantee the delivery 100 %.

Microsoft introduces the WCF (Windows Communication Foundation). The WCF helps reducing the coding complexity. The WCF unifies all coding models for all .NET distributed technologies. Thus, the WCF can help reducing the cost of developing and maintenance distributed applications in the long term.

Both Java and Microsoft technologies have an advantage and a disadvantage. Java distributed technology is only a specification. There are many vendors that implements the specifications including Oracle, BEA, IBM, Sun and Apache. Each vendors provide the same function interfaces but different internal implementations. A Java distributed application may work differently on different implementations. Powerful tools for easily creating Java distributed application from various vendors are not free. Some Java environment tools tend to provide more support on their application servers or their databases. There is no unify environment tool like Visual Studio. Developers' knowledge is scattered on many different implementations and tools. With Microsoft products, developers' knowledge can be better consolidated, because there is only one application server, IIS, and one tool, Visual Studio. There is a free version of Visual Studio called Microsoft Visual Studio Express which is powerful enough to create .NET distributed applications easily. An advantage of using Java distributed technology is a prevention from vendors lock in, because there are more than one vendors offering the solutions.

References

[1] *MSMQ ReferencePoint Suite*. SkillSoft Press, 2002.
[2] Redkar Arohi, Walzer Carlos, Boyd Scot, Costall Richard, Rabold Ken, and Redkar Tejaswi. *Pro MSMQ: Microsoft Message Queue Programming*. Apress, 2004.
[3] Patrick Cauldwell, Rajesh Chawla, Vivek Chopra, Gary Damschen, Chris Dix, Tony Hong, Francis Norton, Uche Ogbuji, Glenn Olander, Mark A. Richman, Kristy Saunders, and Zoran Zaev. *Professional XML Web Services*. Apress, 2004.
[4] Nishith Pathak Chris Peiris. *Pro WCF: Practical Microsoft SOA Implementation*. Apress, 2007.
[5] Bill Evjen, Billy Hollis, Tim McCarthy, Kent Sharkey, and Bill Sheldon. *Professional VB 2005 with .NET 3.0*. Wiley Publishing, Inc., 2007.
[6] Bill Evjen, Kent Sharkey, Thiru Thangarathinam, Michael Kay, Alessandro Vernet, and Sam Ferguson. *Professional XML*. Wiley Publishing, Inc., 2007.
[7] Simon Guest. *Microsoft .NET and J2EE Interoperability Toolkit*. Microsoft Press, 2004.
[8] Judith Hurwitz, Robin Bloor, Carol Baroudi, and Marcia Kaufman. *Service Oriented Architecture For Dummies*. John Wiley & Sons, 2007.
[9] Matthew MacDonald. *Microsoft .NET Distributed Applications: Integrating XML Web Services and .NET Remoting*. Microsoft Press, 2003.
[10] Matthew MacDonald and Mario Szpuszta. *Pro ASP.NET 2.0 in C# 2005*. Apress, 2005.
[11] Shannon Horn Sarah Morgan and Mark Blomsma. *MCTS Self-Paced Training Kit (Exam 70-529): Microsoft .NET Framework 2.0 Distributed Application Development*. Microsoft Press, 2007.
[12] Dominic Selly, Andrew Troelsen, and Tom Barnaby. *Expert ASP.NET 2.0 Advanced Application Design*. Apress, 2006.
[13] Justin Smith. *Inside Microsoft Windows Communication Foundation*. Microsoft Press, 2007.
[14] Rima Patel Sriganesh, Gerald Brose, and Micah Silverman. *Mastering Enterprise JavaBeans 3.0*. Wiley Publishing, Inc., 2006.
[15] Andrew Troelsen. *Pro C# with .NET 3.0, Special Edition*. Apress, 2007.
[16] Paul A. Watters. *Web Services in Finance*. Apress, 2005.
[17] James L. Weaver, Kevin Mukhar, and Jim Crume. *Beginning J2EE 1.4: From Novice to Professional*. Apress, 2004.
[18] Bobby Woolf. *Exploring IBM SOA Technology & Practice*. Maximum Press, 2008.
[19] Kareem Yusuf. *Enterprise Messaging Using JMS and IBM WebSphere*. IBM Press, 2004.