AN ADVANCED TEXT-TO-SPEECH SERVER SYSTEM BASED ON SOAP PROTOCOL

XU Yingying[1], TANG Hao[2] and ZHANG Peiren[1]

[1]Department of Automation
[2]Department of Electronic Engineering and Information Science
University of Science and Technology of China, Hefei, Anhui, 230026, P.R.China
yingerxu@mail.ustc.edu.cn, {tangh, przhang}@ustc.edu.cn

ABSTRACT

The traditional Text-to-Speech (TTS) server system is an interactive service platform for voice generation within an Intranet operating over TCP/IP protocol. In the information era, the ever-expanding Web services require that the service scope of TTS server systems enlarge to the whole Internet. This paper proposes an advanced TTS server system binding with the Simple Object Access Protocol (SOAP) and describes its architecture and implementation. The performance data comparison by experiments is presented in the end of the paper.

1. INTRODUCTION

As organizations try to reinvent themselves as "e-businesses," they find it necessary to offer their customers whole new channels of communication including e-mail, text chat, Web call-back requests, voice over IP, etc. Such real-time Web-based interaction is the key to successful e-commerce initiatives [1]. Within these applications and utilities, the voice application at enterprise level has been attracting more and more attention for not only a new channel to deliver information but also a new business model around the world. The traditional Text-to-Speech (TTS) server system is such an interactive service platform for voice generation within an Intranet operating over TCP/IP protocol, through which the enterprises can centralize the processing of all customer requests for TTS service and provide a new channel of communication in real time.

Figure 1 shows a common framework of the traditional TTS server system. The traditional TTS server system generally employs a client-server architecture. The client side consists of a variety of speech applications, with each application requesting TTS service as its scripting dictates. One or more TTS servers connected via TCP/IP handle these requests, returning voice data corresponding to the text that was handed.

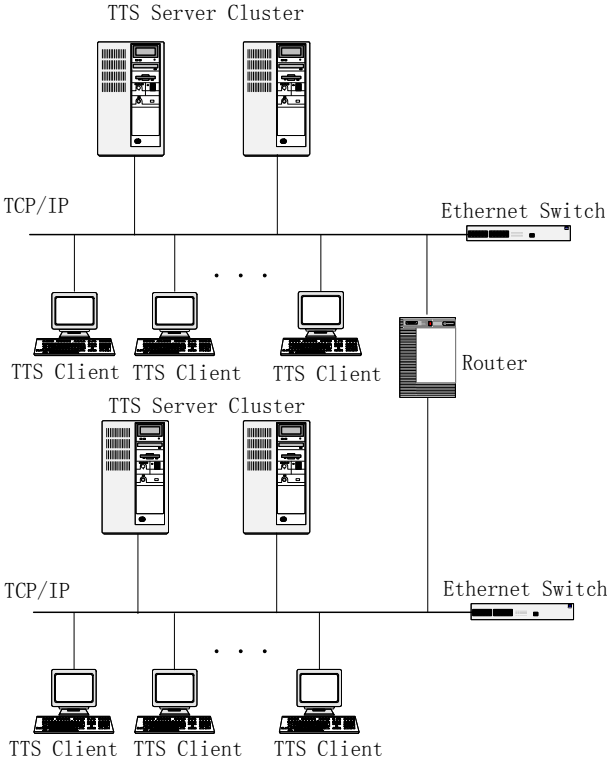

Figure 1. Common framework of traditional TTS server system

As can be seen from Figure 1, the traditional TTS server system runs within the Intranet of a single enterprise. However, in current information era, the ever-expanding Web services urge more and more companies to enter the market offering their products as service solutions. Given these circumstances, it will no longer be sufficient to solely provide the TTS service for an appointed enterprise. Instead, the challenge is to evolve it to Web services to satisfy a broad range of requirements, including all kinds of cross-platform integrations. In order to meet this new challenge and improve customer service, we have developed an advanced TTS server system based on the Simple Object Access Protocol (SOAP).

This paper is organized as follows. Section 2 introduces the SOAP protocol, the reasons why we used SOAP as the underlying infrastructure of our system and the binding fundamentals of SOAP. Section 3 describes the system implementation mechanism and provides the performance data comparison between our system and a traditional TTS server system based on experimental results.

2. INFRASTRUCTURE

SOAP is a way for a program running in one kind of operating system (such as Windows 2000) to communicate with a program in the same or another kind of operating system (such as UNIX) by using HTTP and XML as the mechanisms for information exchange [2]. SOAP supports XML document exchange and provides a convention for Remote Procedure Call (RPC) using XML messages. In fact, SOAP is a platform-independent access protocol.

2.1. Why use SOAP

The use of SOAP in our system as its underlying infrastructure can make communication between client and server objects via HTTP, an application-level protocol for distributed and collaborative hypermedia information systems [3]. The actual network conversation is encoded in XML, a platform-independent and robust makeup language [4]. All these offer tremendous advantages to software-to-software communications.

Because SOAP messages can be carried over the HTTP protocol, they can easily pass through firewalls. Unlike other distributed object models that rely on dynamically assigned ports, SOAP can use HTTP's standard port for transmitting data. SOAP messages can be easily filtered at a firewall because the header information can be mandated to contain meta-information of the interaction that is being made [5]. This saves a firewall application from having to read the entire XML body to manage the request.

Moreover, SOAP is a more easily implementable platform-independent access protocol. SOAP is similar to DCOM and CORBA in that it provides an RPC mechanism for invoking methods remotely. However, SOAP differs in that it is a protocol based on open XML standards and XML document exchange rather than being an object model relying on proprietary binary formats. The SOAP gateway performs a similar function to DCOM and CORBA stubs – translating messages between the SOAP protocol and the language of choice. As a result, SOAP offers vendor, platform and language independence. With SOAP, developers can easily bridge applications written with COM, CORBA or Enterprise JavaBeans™.

2.2. How to bind SOAP

Microsoft SOAP Toolkit provides a wizard that exports the methods of a COM object using SOAP. The COM objects may be written using Visual C++ or Visual Basic. The wizard generates deployment files including the Web Service Description Language (WSDL) file that defines the interface [6].

The WSDL is an XML-based language for describing the network services offered by the server. Essentially, a WSDL document describes how to invoke a service and provides information on the data being exchanged, the sequence of messages for an operation, the protocol bindings and the location of the service [7]. A WSDL document defines services as a collection of endpoints, but separates the abstract definition from the concrete implementation. Messages and port types provide abstract definitions for the data being exchanged and the operations being performed by a service. A binding is provided to map to a port, usually consisting of a URL location and a SOAP binding.

3. SYSTEM IMPLEMENTATION

In our system, we have developed a SOAP-enabled COM object to provide TTS service according to the TTS Application Programming Interface (API). The TTS service can be interpreted by a simple model: First, the client connects to the server on the HTTP default port number. Second, the client sends text to the server through an HTTP POST request. Third, the server processes the request, synthesizes the text and then sends back the generated voice data to the client. Finally, the client invokes a close action to ensure that the connection between the client and the server is destroyed.

To use the COM component to communicate between the TTS server and various clients, we first need to create a WSDL file that describes this COM component on the Web server that runs Microsoft Internet Information Services (IIS). Microsoft SOAP Toolkit provides all the tools to create WSDL files for COM components [8]. Having the WSDL and related listener files on the Web server, the SOAP-enabled COM component becomes a TTS service that the remote clients can access.

3.1. Architecture

The architecture of our client-server based TTS server system is represented in Figure 2. The figure interprets how the objects communicate with one another using Internet connection. Since SOAP is an XML-based protocol, an XML parser is needed on both the client and the server.

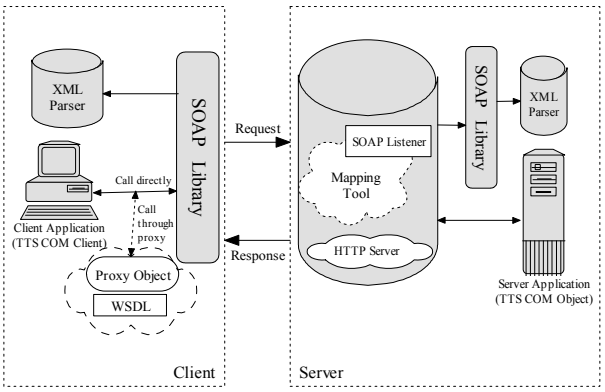

Figure 2. Architecture of our TTS server system

As shown, on the client side, the client application can either call the methods on the server object directly or call through a proxy object to hide all SOAP details. The proxy object is a layer that can make converting existing applications from COM to SOAP easy and give the client application the impression that it is calling the methods on the server object. The proxy object is responsible for serializing the parameters, generating SOAP messages, parsing and de-serializing the response messages and returning the native data types. Figure 3 shows the code snippet to initialize the proxy object.

```
CString Path;
GetModuleFileName(AfxGetApp()->m_hInstance,
Path.GetBuffer(_MAX_PATH), _MAX_PATH);
Path.ReleaseBuffer();

Path = Path.Left(Path.ReverseFind(_T('\\'))+1);

CString WsdlFile = Path + _T("TTSService.wsdl");
CString WsmlFile = Path + _T("TTSService.wsml");

try
{
   m_TTSProxy.Initialize(WsdlFile, WsmlFile);
}
catch( _com_error Error )
{
   DisplayError(_T("Cannot initialize TTS Service proxy"), Error);
   EndDialog(0);
   return TRUE;
}
```

Figure 3. Initialization code of proxy object

On the server side, the SOAP listener takes charge of reading the incoming SOAP request, building a SOAP response message that contains the result of the operation and sends this message to the client. We can use different mapping tools to map SOAP requests in any format we like as long as both the client and the server agree on it. The SOAP messages are wrapped according to the methods that the TTS COM object provides:

- TTSConnect - Connects to the TTS service and gets a handle <H> identifying the session
- TTSDisconnect - Destroys the session identified by <H>
- TTSSynthText - Requests that TTS service accept the text and return a status code <S>
- TTSFetchnext - <S>=1 indicates that there are still more voice data available on the server that can be fetched by continuously invoking this method. This method also returns a status code <S> representing the same meaning as it does in TTSSynthText
- TTSSetParameters - Sets synthesis parameters such as speaking rate, pitch, volume, etc.
- TTSGetParameters - Gets synthesis parameters
- TTSClean - Gets rid of the remaining voice data on the server if a normal session is interrupted

3.2. WSDL description

WSDL provides an interface definition language for SOAP. From Figure 4, we can see the components of a WSDL document and their relationships.

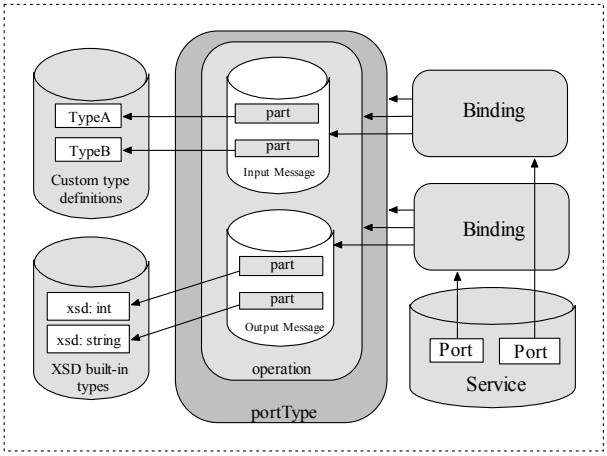

Figure 4. Components of WSDL document

In our system, we defined a TTSData complex type for the message part. The message part describes the logical contents of the messages being communicated. Figure 5 shows the SynthText request and corresponding response message.

```
<message name='TTS.SynthText'>
   <part name='usertext' type='xsd:string'/>
   <part name='pbIsSynthText' type='xsd:boolean'/>
</message>
<message name='TTS.SynthTextResponse'>
   <part name='result' type='xsd:TTSData' />
</message>
```

Figure 5. SynthText request and response message

In the portType part, a set of operations and the messages involved with each of the operations are described like this:

```
<portType name='TTSSoapPort'>
    <operation name='SynthText'
              parameterOrder='usertext pbIsSynthText'
      <input message='wsdlns:TTS.SynthText' />
      <output message='wsdlns:TTS.SynthTextResponse' /
    </operation>
    …
</portType>
```

Figure 6. portType part in WSDL document

When a client sends a SOAP request to a TTS server requesting an operation, it must identify the service, a port in the service and the operation it wants to execute along with the input parameter values. The following code fragment shows the service element.

```
<service name='TTSservice' >
 <port name='TTSSoapPort' binding='wsdlns:TTSSoapBinding'>
   <soap:address location='http://MSSoapTTSServer/
            TTSService/Res/TTSservice.WSDL' />
 </port>
</service>
```

Figure 7. Service part in WSDL document

3.3. Performance comparison

To test the performance of our newly established TTS server system, we have performed experiments on it as well as on a traditional TTS server system that shares the same TTS engine. Each of the two systems ran on a P-III 550 server with 256M RAM. We set up 30 clients (ordinary PCs) connected to the server via 100M Ethernet. In our tests, we measured the durations between the time when the clients began to send text to the server and the time when they began to receive the voice data. The text materials used include 1000 20-byte-or-so sentences, 500 800-byte-or-so poems and 200 5000-byte-or-so articles consisting of many kinds of symbols like numerals, punctuations, English words, etc. Table 1 contains the performance data (30-way average time) of the two systems according to different kind of text.

System	Sentences	Poems	Articles
Using TCP/IP	4.705	2.462	2.970
Binding SOAP	5.710	2.979	3.364

Table 1. Response time (in second) of two systems

Contrast to the traditional TTS server system using TCP/IP, the delay of our system is about 400ms more than that of the traditional system on average. Considering that the actual data transfer time is much less than the voice generation time on the server, we conclude that by binding SOAP we haven't degraded system performance.

4. CONCLUSION

The best features of SOAP are the clarity of the messages, the firewall passing capability and the easy RPC support. Based on SOAP protocol, our TTS server system can provide TTS service in the Internet with improved interoperability. To embrace different client systems, we can use different mapping tools to map SOAP requests within ASP, JSP, CGI, SERVLET, etc. In the future, our research will focus on conducting data compression within the SOAP protocol and further enhancing the flexibility and reliability of our TTS server system.

5. REFERENCES

[1] Donald E.Brown, "The Interaction Center Platform White Paper," http://www.inin.com.

[2] "A Definition of Web Services," http://www.integra.net.uk /services, July 2002.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners, Lee, "Network Working Group Request for Comments: 2068," http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc2068.html, January 1997.

[4] Timothy M. Chester, "Cross-Platform Integration with XML and SOAP," *IT Professional, IEEE*, Volume: 3 Issue: 5, Page(s): 26 -34, Sept.-Oct. 2001.

[5] Yasser Shohoud, *Learn XML Web Services Development*, http://www.learnxmlws.com/book/chapters.

[6] Rob Caron, "Develop a Web Service: Up and Running with the SOAP Toolkit for Visual Studio," *MSDN Magazine*, http://msdn.microsoft.com/library, Aug. 2000.

[7] Microsoft Corporation, "Help of SOAP Toolkit 3.0 Beta 1," http://msdn.microsoft.com/downloads, 2002.

[8] Microsoft SOAP Toolkit, http://msdn.microsoft.com/xml /default.asp.