STUDY ON DISTRIBUTED SPEECH SYNTHESIS SYSTEM

TANG Hao, YIN Bo and WANG Ren-Hua

Department of Electronic Engineering and Information Science
University of Science and Technology of China, Hefei, Anhui, 230026, P.R.China
tangh@ustc.edu.cn, boyin@iflytek.com and rhw@ustc.edu.cn

ABSTRACT

Distributed Speech Synthesis (DSS) is a novel field of study and will be playing an active role in the domain of speech technology in the post-PC era. This paper focuses on the architecture design, implementation mechanism and optimization considerations of DSS systems. First, client-server based DSS architectures are investigated in server, client and network parts, respectively. Second, related system implementation issues are discussed correspondingly. Finally, system optimization suggestions in the aspects of data security, QoS control and load balancing between DSS server and client through dynamic arbitration are presented.

1. INTRODUCTION

Along with the ever-expanding Web services and increasingly rising wired and wireless data businesses, there have been many research and engineering efforts in developing the speech-enabled applications over the multimedia networks. Considering the diversified and distributed characteristics of the multimedia networks, it is perfect to implement the speech-enabled applications over such service networks with distributed architectures. Unlike its counterpart DSR (Distributed Speech Recognition), which has been studied carefully and has been standardized by the ETSI STQ Aurora DSR Working Group [1], DSS is a novel field of study and will be playing an active role in the domain of speech technology in the post-PC era.

DSS involves various technologies including text-to-speech synthesis, distributed computing and system design, network transmissions and protocols, multithreaded programming, XML and Web techniques, which necessitates a detailed and thorough study on DSS.

Fig. 1 represents the principal framework of DSS. In the service network, DSS clients are a variety of speech-enabled devices including multimedia PCs, wired and wireless telephones, PDAs and many other personal terminals. One or more DSS servers are connected to the backbone of the service network through a Load Balancing Proxy (LBP). As service starts, DSS clients send text to the LBP. The LBP dispatches the text to one of the DSS servers with a minimum load. The DSS server accepts the text, converts it into prosodic features and sends the feature data to the appropriate DSS client for ultimate processing.

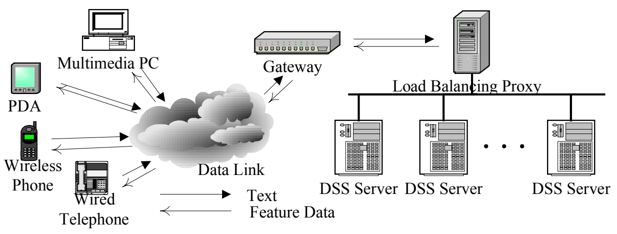

Fig. 1 Principal framework of DSS

2. ARCHITECTURES OF DSS

2.1. Fundamental architecture of DSS

Current speech synthesis techniques fall into two distinct trends: either becoming increasingly complicated in order to produce highly natural speech (e.g., the corpus-based approach) or otherwise simplistic in order to meet the limitations required by low-end computing devices. The latter, however, always results in unsatisfying naturalness.

Based on traditional client-only or server-only architecture, realization of high-quality speech synthesis on personal terminals is infeasible. The reasons lie in 1) limited CPU and memory resource available on personal terminals which restricts them to performing non-complicated tasks and 2) low bandwidth and high error rate of wireless networks which prevent them from transmitting pure voice. Client-server based DSS architecture makes it possible to obtain highly natural speech on resource-sensitive personal terminals by demanding that heavy work such as linguistic and prosodic processing be done at server side.

The fundamental architecture of DSS is illustrated in Fig. 2. It consists of three interactive parts: server, client and network. In this client-server architecture, DSS server is designated with linguistic and prosodic processing known simply as the most resource-demanding portion of text-to-speech synthesis, while DSS client performs a relatively simple task, the ultimate speech synthesis. The intermediate results carrying prosodic features generated by DSS server are transmitted to DSS client via proper underlying networks and protocols. To-be-synthesized content can be sought from many sources throughout the Internet, e.g., an independent content server. It follows that two sorts of requests can be made by DSS client to DSS server: one is the Content Request followed by the actual content (usually text) and the other is the URL Request requiring that DSS server download the content specified by the URL provided.

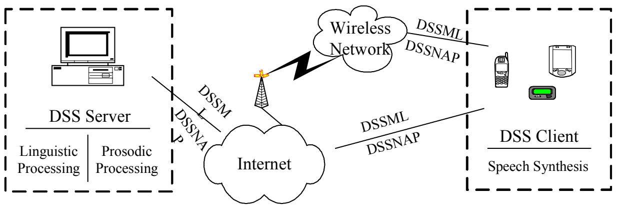

Fig. 2 Fundamental architecture of DSS

2.2. Architecture of DSS server

DSS server is the entity that accomplishes linguistic and prosodic processing and then sends the results to DSS client according to certain concerted protocol and data exchange standard. The architecture of DSS server is illustrated in Fig. 3. Four components, the core engine, the server browser, the transcoder and the XML generator constitute DSS server. In response to the requests made from DSS client, DSS server either extracts the content (Content Request) or drives its server browser component to download the content from the Internet (URL Request). The expected content can be plaintext, HTML, E-mail or other formatted text. The content must be passed to the transcoder that translates it into plaintext understandable to the core engine. And then in the core engine, the plaintext is converted into prosodic features, which proceed to be processed by the XML generator to generate their XML representation before being sent to DSS client.

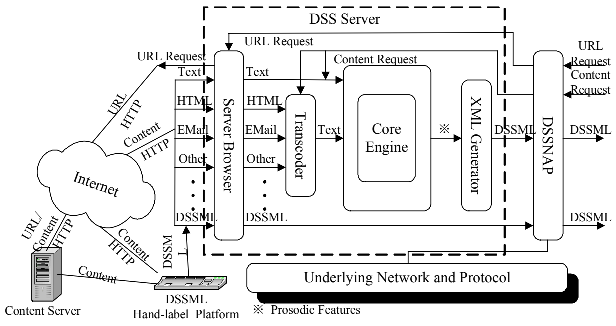

Fig. 3 Architecture of DSS server

XML documents representing pre-generated prosodic features will be passed directly to DSS client if requested. These documents are constructed by an offline DSS server and if necessary, adjusted manually before they reach DSS server.

2.3. Architecture of DSS client

DSS client is the entity that converts the prosodic features received into speech. It observes the same protocol and data exchange standard as DSS server. As shown in Fig. 4, DSS client comprises the core engine, the XML parser and the client API that is invoked by the applications whenever needed. A particular application captures the content from the Internet and then passes it to the client API, or instead passes the URL indicating where the content is located. In the former case, DSS client makes a Content Request to DSS server, while in the latter case makes a URL Request. On success, the XML document returned is firstly processed by the XML parser to restore the original prosodic features, which are then conveyed to the core engine for ultimate speech synthesis. In addition to the speech outputted to the application, requests for external data initiated by DSS client are sent through the client API. Thus, the client API must realize callback mechanisms and force the applications to response to such requests.

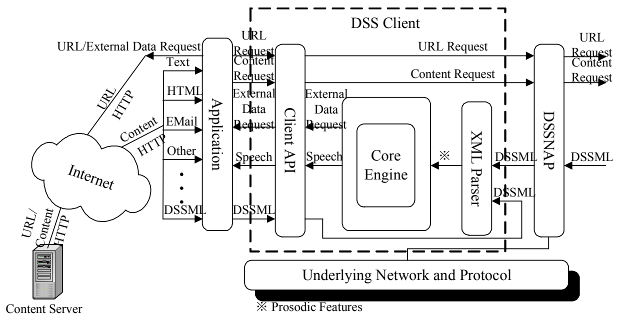

Fig. 4 Architecture of DSS client

In case that the initial content is an XML document representing pre-generated prosodic features, the content will be sent to the XML parser of DSS client instead of being sent to DSS server.

2.4. Networks and protocols

The physical connections between DSS server and DSS client vary according to different application environments. However, we have defined a uniform high-layer DSS Network Application Protocol (DSSNAP) to hide the diversity of underlying protocols and networks from the users. DSSNAP realizes features of distributed systems as follows: 1) DSS Remote Procedure Call (RPC) mechanism; 2) Load balance; 3) Scalability; 4) Fault tolerance and 5)

Data compression [2]. The protocol stack of DSS is shown in Fig. 5.

DSSNMP: Distributed Speech Synthesis Network Management Protocol
DSSNAP: Distributed Speech Synthesis Network Application Protocol

DSSNAP	DSSNMP	DSSNAP
HTTP	SNMP	HTTP
TCP	TCP/UDP	TCP
IP	IP	IP
Physical Link	Physical Link	Physical Link
Server Note	Intermediate Note	Client Note

Fig. 5 Protocol stack of DSS

Data exchange plays a vital role in DSS architectures and therefore must be standardized. A key consideration is the hierarchical representation of prosodic features, which can be understood not only by all kinds of DSS clients built with different techniques but also by human readers, even by those with little knowledge of phonetics. High modularization of text-to-speech synthesis as well as the proved XML techniques make it possible, though not easy, to accomplish this mission. An XML-based markup language, called DSS Markup Language (DSSML), has been defined to act as the proposed standard of data exchange in DSS systems. In contrast to the standards of VoiceXML and SALT, DSSML focuses on the hierarchical representation of prosodic features of the content and is to some extent similar to existing markups such as SSML, STML, JSML and SABLE [3].

3. IMPLEMENTATION MECHANISM OF DSS SYSTEMS

Building a DSS system can be a very complex task, as it requires multithread and multi-user support at the server side, limited computing cost and storage at the client side and low network transmission rate. In this section we will discuss critical issues concerning these problems.

3.1. Parallel DSS server

In the service network, DSS servers provide concurrent service to DSS clients. When the workload exceeds the capabilities of the DSS servers, additional hardware should be added. Such scalability feature is supported by one of the two primary methods, both standing out as mature, broadly effective technologies:

- Large-scale symmetric multiprocessing (SMP): SMP increases system capability by building larger SMP machines: 8 CPUs, 16 CPUs, 32 CPUs, and so on. The primary benefit is that application software does not need to change - the operating system does the work. In addition, maintaining one system is easier than maintaining a bunch of smaller systems. However, SMP has many hurdles to overcome. First, large-scale SMP machines are expensive. Second, many operating systems such as NT have flawed thread schedulers and RAM limits that restrict the amount of CPUs available. Third, SMP does not guarantee availability and fault tolerance.
- Clustering: Clustering lets you link separate computers (or nodes) to work as a single system. If one of the two clustered servers fails, the functioning server picks up the load. In addition to high availability, clustering helps achieve high performance. However, because clusters have looser processor-to-processor communications than SMP systems, more care must be taken to structure their workloads for scalable performance.

Another critical issue regarding DSS server is the internal parallel and multithreaded program. Here we propose two approaches. One is to use the standardized distributed object computing technologies such as CORBA, DCOM and Microsoft .Net. The other is to use SOAP (Simple Object Access Protocol). Many technical details of these technologies can be found in [3], [4], [5].

3.2. Low-cost DSS client

In general, DSS clients are those personal terminals such as PDAs and cell phones with limited computing and storage capabilities. This prevents them from performing tasks that consume lots of resources. A DSS client needs a speech database whose size can be as large as several hundred megabytes (16k, 16bits, pcm). However, for most DSS clients, the size of the speech database should be constrained to 10 megabytes or less.

A solution to this problem is to employ one of the speech coding algorithms to compress the speech database. A wide range of speech coding algorithms standardized by the International Telecommunications Union (ITU) are available (such as G.721, G.723, G.728, G.729, etc.). They span the bitrate from 2.4 kbits/s to 64 kbits/s. Thus, the best compression rate achieved can be up to 1/100.

If the modern corpus-based speech synthesis technique is used in DSS systems, the speech database is far larger, e.g., 2-3 gigabytes. It is impossible to satisfy the storage limitation of DSS clients if we solely rely on speech coding algorithms. Recently, the Speech Lab at University of Science and Technology of China has proposed a scalable algorithm for tailoring the speech database [6]. With this algorithm, researchers have successfully reduced the size of a speech database from 2 gigabytes to 100 megabytes. However, the loss of speech naturalness is tiny and thus can be neglected.

In practice, the above two solutions are jointly used to achieve best compression rate.

3.3. XML data compression

A data format for structured document interchange on the Internet, XML has many excellent features such as human/machine readability, platform independence and extensive compatibility. However, because XML contains extra structural information besides the original data, data redundancy is inevitable and will affect data transmission rate if network bandwidth is critical. XML-based DSSML documents act as data exchange for DSS systems. Thus, they need to be compressed before being transmitted on the network for the efficient use of network bandwidth.

Many a researcher has conducted in-depth research on XML compression. Alain Trottier introduced an approach called "zxml". Zxml simply eliminates duplicate/redundant tags and adds a header and template area to lists tag names and values.

Xmill, which was developed by the University of PA and AT&T Labs, groups data items according to their elements. Each group is then compressed separately via gzip.

XMLZip reduces the size of XML files while retaining the accessibility of the DOM API, allowing applications to access data in its compressed form.

WAP Binary XML (WBXML) defines a compact binary representation of XML for transmission of WML content. WBXML preserves element structure but the encoding process removes DOCTYPEs, Comments, INCLUDE/IGNORE sections.

Millau defines an encoding format that extends WBXML. Millau preserves the XML structure and compresses the data separately.

We have conducted experiments to test the speed and compression rate of the above algorithms. The results indicate that Xmill obtains the best compression rate while zxml obtains the best speed. In DSS systems, the selection of XML compression algorithm depends on practical requirements.

4. OPTIMIZATION CONSIDERATIONS OF DSS SYSTEMS

A robust DSS system should be optimized to meet different practical requirements. In this section we will make several optimization suggestions for future considerations.

4.1. Data security

Data exchange between DSS servers and clients is in the open XML format. This may cause big problems in security. A number of security technologies including authentication protocol, encryption algorithm and digital signature can be employed in implementing DSS systems. However, tradeoffs between the security level and system performance should be seriously considered.

4.2. QoS control

QoS (Quality of Service) guarantees low delay, low error rate and high throughput of DSS systems. We recommend that DSS systems define several QoS parameters, based on which the underlying TCP/IP protocol can negotiate with RSVP, MPLS or RTP to establish the optimal route for connection. Once the connection is established, the expected bandwidth can be obtained.

4.3. Load balancing between DSS server and client

The LBP handles load balancing among DSS servers. We may need a different mechanism to handle the load balancing between DSS server and client. DSSML is a hierarchical representation of prosodic features. Different layers of DSSML contain sufficient information for DSS clients but require different computing capabilities on servers and clients. We can determine in real time which layer of DSSML is to be used with a dynamic arbitration algorithm. Main consideration factors are the computing capability of individual client, the bandwidth of currently established connection and the sever workload.

5. CONCLUSION

This paper presents a comprehensive study on DSS. Based on client-server model, DSS architectures are detailed in server, client and network parts. Feasible implementation mechanism and future optimization considerations of DSS systems are provided. DSS is a novel field of study and a challenging work involving many research and engineering topics. Our future research will focus on arbitration algorithms for load balancing.

6. REFERENCES

[1] WeiQi Zhang et al, "The Study on Distributed Speech Recognition System," *in Proc. 25th IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 3, pp. 1431-1434, 2000.

[2] Jie Wu, *Distributed System Design*, CRC Press, Boca Raton, FL, 1998.

[3] The World Wide Web Consortium, http://www.w3.org.

[4] P. Emerald Chung et al, "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer," *in C++ Report*, Jan. 1998, http://www.research.microsoft.com/~ymwang/papers/HTML/DCOMnCORBA/S.html.

[5] Homepage for Microsoft .Net, http://www.microsoft.com/net.

[6] Zhi-Wei Shuang, Yu Hu, Zhen-Hua Ling and Ren-Hua Wang, "A Miniature Chinese TTS System Based on Tailored Corpus," *in Proc. 7th Int. Conf. Spoken Lang., ICSLP 2002*, pp. 2389–2392, 2002.