1. INTRODUCTION

In 1995, in order to make speech-processing technology available to personal computers, Bell Labs (then within AT&T) began work on a software development kit for automatic speech recognition (ASR) and text-to-speech synthesis (TTS). The product, named Watson™ [2], has a standard API and runs on personal computers (PC). Since Watson’s inception, Bell Labs, now part of Lucent Technologies, has continued to research and develop software-based speech solutions. This effort leads to the Lucent speech software product LSAP™ – Lucent Speech Application Platform. LSAP, like Watson before it, LSAP combines ASR and TTS functionality under a single product.

Compare to hardware based speech solutions, or system based on proprietary interfaces, the main advantages of software-based speech solutions with MS SAPI are:

- **Easy to use:** No special hardware is needed. The start up cost is lower than hardware or proprietary system solutions. A common MS Windows™ (NT/9x/2000) PC with multimedia support is sufficient. On the software development side, since a MS SAPI speech engine can be presented as a DCOM (Distributed Component Object Model), or an ActiveX object, it is easy to plug it into MS Windows applications.

- **Rich application integration support:** Most software development tools on Windows support MS SAPI, since it is in DCOM and ActiveX specifications. Also there is a rich set of software modules that are easy to be integrated with speech engines for back end applications, such as database engines, messaging systems, office tools, etc.

- **Flexible:** A software speech system is easy to upgrade. It is also easy to modify to fit different customer’s needs.

Thanks to the recent PC hardware performance advancement, and the availability of low cost symmetric multi processor (SMP) PCs, we saw the opportunity on using MS SAPI speech engines to provide speech processing for service applications. This does not only inherit the advantages listed above, but also gives our new competitive advantages:

- **The general purpose SMP PC performance improvement is much faster than a proprietary hardware.** And there are more and more QoS (quality of service) and resource management software available as standard components on PC operating systems. Therefore, it is cost effective to develop and deploy software speech solutions on PC operating systems.

- **It is easy for IP Telephony integration.** We only need to map IP voice devices as SAPI speech engine’s audio objects for Voice over IP (VoIP) speech audio input/output. Under MS Windows TAPI 3.0 [3], a VoIP device is mapped with a behavior similar to a traditional telephone device.

- **For a standard speech API,** we can develop a single engine software system which address customers from a single desktop user to distributed network services.

In 1998, we started development of new versions of the speech engines that comprised LSAP to address service application and scalability issues. To simplify software development process, and to make it more flexible to address our customer’s needs, we decided to separate ASR and TTS engines as two software packages, but with option to integrate them together. This development leads new version of Lucent software speech engines: LASR3 [4] and Lucent Text-To-Speech Version 3 (LTTS3) [5].

This paper discusses the LASR3 algorithms, features, and architecture design considerations for scalable, multi-threaded speech recognition.

2. LASR3 FEATURES

In LASR3, we used the Bell Labs Automatic Speech Recognition (BLASR) algorithm set [6, 7, 8] as the engine core, which significantly improved performance on large vocabulary and natural language speech recognition.
In order to provide high accuracy speech recognition, LASR3 possesses a unique feature set that includes the following:

**Speaker Independence speech recognition engine**

The LASR3 engine is a speaker independent system. Meaning, a user does not need to train the engine before using it. This feature is necessary for service application, where we do not have prior knowledge of the users.

LASR3 inherits BLASR’s multi-layer network topology (word, phoneme, and acoustic networks)[7, 8]. Based on network building and decoding strategies, LASR3 engine may be configured to the following three modes:

- **Static decoder:** this is an implementation of BLASR frame-synchronous Viterbi beam search decoder (e.g. [6]). It uses precompiled decoding graph with tri-phone context dependent acoustic modeling for high-resolution speech recognition. For many service applications, most of the dialogue turns needs only fixed speech recognition tasks with small vocabulary (such as menu item selection, yes/no, etc.). This decoder mode can be used to preload many of such small tasks to reduce system dynamics and lower decoder graph building overhead.

- **WAVE decoder:** this is a frame-synchronous Viterbi decoder that adjusts (expand and release) its decoding graph at run-time, based on search demand [7]. With little computation overhead, the decoding network size could be reduced to 5%-10% of the static decoder. An arc prediction algorithm is used with WAVE decoder to further reduce decoder graph expansion. The reduced decoder footprint makes it more suitable for resource constraint environment, such as concurrent multi-channel speech recognition-server, or runs on a single board computer.

- **NhWAVE decoder** (WAVE2): The decoder combines WAVE decoder with quasi-stack decoding strategy [8]. It is more efficient (both speed and memory usage) for very large vocabulary speech recognition tasks with long-span statistical language models.

In addition, LASR3 implements BLASR’s high-resolution N-best decoder [9] to provide multiple recognition candidates.

**Acoustic Modeling**

Current LASR3 release supports two languages: US English and Mexican Spanish. Acoustic models for other languages can be added in the future, without changing the LASR engine.

LASR3 shares the same acoustic model format BL_FHMM (Bell Labs Flexible HMM format) and model manager module with our research software. This enables the LASR engine to use the high-resolution decision tree based models developed in Bell Labs research for large vocabulary speech recognition [10]. For the basic speech application coverage, we provide a general-purpose model that combined context dependent connected digit models with general sub-word phone models to cover most speech recognition tasks with high accuracy. Since a large portion of the model training data were from live telephony network data collection and from field trial of speech applications, the models we provided are robust to real world noise environment. Models trained by other speech software such as HTK toolkit can be converted to the LASR3 model format as well.

**Front-End Signal Processing**

LASR3 front-end is model driven: The comprehensive BL_FHMM header provides sufficient front-end signal processing configuration during model building. And LASR3 uses these parameters to configure feature extraction module. This reduces the probability of front-end configuration mismatch with acoustic model building condition. The current LASR3 front end supports both LPC cepstrum and mel-scaled cepstrum feature extractions. We provide configuration interface to override the model parameter as well. To address the diversity of speech input, which ranges from desktop microphones to wireless telephone handsets, we developed a look-ahead adaptive channel equalization algorithm [13] and a robust speech end pointing algorithm [14] which adjust their parameters dynamically to fit noisy acoustic environment at run time.

**Grammars**

LASR3 supports SAPI context free grammar (CFG) type, with grammar dynamics support (i.e., import rules, and dynamic list, the active decoder will do dynamic re-linking when a grammar runtime dynamic happens.). The software also includes a command line CFG grammar compiler. An application can also use third party grammar tools, which generate SAPI 4.0 compliant binary grammars. To make LASR3 internet speech processing ready, along with MS SAPI grammar support, we developed a XML grammar format [4]. A speech recognition grammar written in this format can be loaded directly from internet URL to LASR3 engine to construct decoding network on the fly. With little effort, this XML interface can be conformed to the future VoiceXML [12] speech grammar standard.

**Dictionary**

To address application demand for multi-lingual speech recognition, as a MS SAPI compatible speech recognizer, LASR3 accepts Unicode text and phoneme representations. It delivers US English and Mexican Spanish in one package, with built-in rule based lexicon dictionary support. Although LASR3 provides a built-in dictionary to transcribe words to phoneme/sub word sequences, its architecture allows application-specified dictionary to supersede a default lexicon for customized pronunciations. This is useful for applications that need special pronunciation control, such as name dialing with non-native names, and multiple pronunciations of a name. We also provide tools to help user dictionary building.

**Recognition Confidence Measure and Rejection**

Filler model based confidence measure [11] is built-in LASR3 decoder to provide confidence scores along with recognition results.
3. ARCHITECTURE DESIGN

The main focus of LASR3’s design is to make it scalable for concurrent, multi-channel speech recognition, with high recognition accuracy and real-time response.

Different from previous LSAP versions, we used new Bell Labs Automatic Speech Recognition (BLASR) as the engine core, which significantly improved performance on large vocabulary and natural language speech recognition. It is also more suitable to make it thread safe for multi-channel scaling. To make the software close to most advanced speech research result and to reduce the development and testing cost, we inherited most of the source code written in the research BLASR version. We only made necessary changes to support LASR3 requirements on object oriented architecture, multi-threading, streaming audio input, and SAPI grammar support, etc.

During LASR3 development, the goal was to develop a service quality, multi-lingual, speaker independent, scalable multi-channel speech recognition server that could be used in telephony and internet speech services. We also wanted to maintain the software reusability across research and LASR3 production codes. Therefore, we adopted a top-down object-oriented software component technology, and wrap speech recognition algorithm code in libraries for each engine components, then we add DCOM interface on top of it. This methodology effectively separated interface with algorithm implementation that makes it easy to port LASR3 engine to other APIs. The algorithm library code is developed to be platform independent and reusable among researchers. LASR3 provides a code base between research and development for easy future enhancement.

During LASR3 development, the goal was to develop a service quality, multi-lingual, speaker independent, scalable multi-channel speech recognition server that could be used in telephony and internet speech services. We also wanted to maintain the software reusability across research and LASR3 production codes. Therefore, we adopted a top-down object-oriented software component technology, and wrap speech recognition algorithm code in libraries for each engine components, then we add DCOM interface on top of it. This methodology effectively separated interface with algorithm implementation that makes it easy to port LASR3 engine to other APIs. The algorithm library code is developed to be platform independent and reusable among researchers. LASR3 provides a code base between research and development for easy future enhancement.

To make LASR3 scalable from desktop single user system to speech recognition server environment distributed on a network, we designed the LASR3 system with a mixed threading model on the MS Windows environment: the apartment thread model to make LASR 3 interface SAPI compatible; the underline free thread model to make LASR3 engine a highly scalable and efficient on parallel processing or multi-processor systems. At LASR3 run time, the recognizer front-end audio input signal processing and the backend decoder engine are running in separated threads, and each of the recognition channels runs in its own set of front-end – backend thread set. Asynchronous procedure call and message passing are used to coordinate the system process. This threading model delivers a maximum efficiency on multi-processor systems. This model may also be applied to other modern operating system with proper thread support.

LASR3 was designed as a set of DCOM components. We partition the engine into two main pieces: front-end and backend. The front end includes MS SAPI compatibility layer and signal processing components. The backend includes decoder, grammar factory and manager, model manager, lexicon dictionary manager, etc. The backend server may run as a remote server to provide speech recognition service through SAPI DCOM client proxy, or in-process mode (where the application links LASR3 dynamic link libraries (DLL) in process) for desktop applications. Figure 1 shows LASR3 component architecture. Here we discuss the main modules in LASR3.

Front-end module (frontend.dll)

LASR3 front-end module includes speech signal processing functions such as feature extraction, adaptive channel equalization, and speech end pointing (described in last section). It also provides a SAPI compatibility layer to an application client uses SAPI interface. Since LASR3 front-end is a lightweight signal processing module, it may reside at client device such as a PDA, or a mobile phone set for better near end signal processing adaptation. The speech end-pointer keeps monitoring the input audio to identify speech segments. Only in speech audio segments compressed by feature extraction are sent to backend engine side to reduce the data traffic. This method is suitable for distributed internet speech processing applications.

Decoder module (decoobj.dll)

This module is the master functional module at the LASR3 backend. Each of the speech recognition instance creation request creates a decoder instance, based on the requested decoder mode, which specifies language, input channel (microphone, or telephone, etc.), and acoustic modeling (general, or high accuracy context dependent models for large vocabulary speech recognition). The decoder instance initializes itself based on mode specification to set up signal processing controls, acoustic model object, and lexicon manager to for the language specified. When the SAPI client request a recognition task by sending a grammar object to the decoder, it will translate the SAPI grammar object to LASR3 internal decoding network by the network manager (embedded in decoobj.dll), based on the decoder mode selection described in Section 2. The network manager uses information provided by the model manager (mdelmgr.dll) to connect a symbolic decoding network (phone network) to a corresponding acoustic decoding network (decoding graph). When the grammar is activated, the decoder thread starts processing the speech feature vectors sent by the front-end. When it recognizes a text string that fits in grammar,
a proper event is generated to notify the application with the grammar object. On retrieving the recognition results, it generates the result object to send it to the application. For different decoder channels, different decoder modes and grammars may be used concurrently in different thread sets.

**Model manager module (modelmgr.dll)**

This module provides acoustic model management service to other LASR3 engine components. It encapsulates the acoustic model parameter detail and provides model information, parameter values to share with all LASR3 channels. To make it efficient on multi-channel sharing, we used a client-server model in the model manager. At the server side, the model manager loads model file as a read only sharable memory map, then provides a low level model mapping to provide the entire model object to a model manager client. At the client side, each model manager client derives a second model mapping which is usable for likelihood computation, etc. This second mapping contains private data members used by the client to write information such as score caching, etc. With little overhead, this two level model mapping design provides an efficient way to share acoustic models among multiple recognition channels.

**Grammar manager module (grmmgr.dll)**

All SAPI binary grammar objects are under the grammar manager to provide grammar object management (add, delete, etc.) dynamic sharing (import rules, dynamic lists). A built-in grammar-processing engine is responsible to parse SAPI grammar objects, create phone graph by using word lexicon information provided by the lexicon manager. LASR3 also provides runtime XML based text grammar parsing, assisted by the Microsoft XML parser (part of MS Internet Explorer 5).

**Lexicon manager module (lexicon.dll)**

The LASR3 lexicon manager provides a hierarchical word to lexicon transcription strategy [15] to LASR3 grammar manager:

If the word needs to transcribe is in LASR3 runtime dictionary cache, the lexicon manager will provides this cached result as the lexical query return.

If the word is not in cache, the lexicon manager looks for the word in a user dictionary (can be modified by users and rebuilt by using Lucent Custom Dictionary Editor (CDE)). If it finds the word, it returns the result and caches it for future reference.

If the word is neither in cache nor in user dictionary, the lexicon manager calls a query interface to the LTTS3 lexicon engine (bundled with LASR3 distribution) to use LTTS rule based text analysis to provide the pronunciation of the word. The result is cached for future use as well.

This transcription strategy provides an efficient, flexible lexicon runtime transcription service, with practically no vocabulary limitation.

4. **SUMMARY**

Based upon decades of Bell Labs speech processing technology, the LASR3 engine possesses high recognition accuracy and good scalability, which makes it usable in a wide range of speech service applications. The software engineering methodology applied in LASR3 makes the engine highly reusable, and its flexible, object oriented component architecture makes it easy to add new features.

5. **ACKNOWLEDGEMENTS**

The authors wish to thank Dr. Sungbok Lee and Dr. Xiaoming Weng for their work on LASR3 testing and QA process development, and to Mr. Vadim Vinshtok for his help with the LASR3 architecture design and LASR3 software development.

6. **REFERENCES**