SPOKEN LANGUAGE INTERFACE IN ECMA/ISO TELECOMMUNICATION STANDARDS

Kuansan Wang

Speech Technology Group, Microsoft Corporation
One Microsoft Way, Redmond, WA 98052, USA
Phone: 1(425)703-8377, FAX: 1(425)93-MSFAX

ABSTRACT

Computer Supported Telecommunication Applications (CSTA) is a widely adopted ECMA/ISO standard suite for global and enterprise communications. As it becomes evident that spoken language systems are playing an ever important role in telecommunications, CSTA has been revised to include a specification on spoken language interfaces so as to promote the interoperability and adoption of spoken language technologies. The new specification shares the same design philosophy as the Speech Application Language Tags, or SALT, that provides simple yet rich functionality for spoken language processing. The tight integration of speech into the telecommunication infrastructure not only presents many technical benefits but also ushers in an era that can potentially make spoken language technologies even more prevalent in our daily lives. This paper describes the design considerations for the specification, and shows how this new standard can be used in conjunction with other standards to create powerful speech applications.

1. INTRODUCTION

Computer Supported Telecommunication Applications (CSTA) refers to a collection of Ecma and ISO standards that govern the telecommunication operations and the interoperability of the underlying infrastructure. Both Ecma and ISO are renowned standard setting organizations. Among the most influential and widely known standards they publish are the coding formats for compact and digital versatile disks, as well as the ECMAScript programming language (ECMA-262, also known as Jscript or JavaScript) [1] that, together with HTML [2], becomes the *lingua franca* of the World Wide Web. Widely implemented by the top telecommunication equipment manufacturers, CSTA plays the similar pivotal role in insuring communication links crossing equipment boundaries can be established smoothly and transparently with high quality.

As technology improves and becomes affordable, more and more spoken language systems are being deployed to enhance communications. From personal voice activated dialing on the mobile devices, telephony self service systems using interactive voice responses (IVR), to enterprise call center automation based on sophisticated computer telephony integration, the role of spoken language technology is becoming ever prominent and important. The technical group steering the CSTA standards has recognized this trend and steadily revised and amended the specification to accommodate the new technologies. In terms of speech related technologies, for example, the group published a Voice Browser Profile (ECMA-TR/85) [1] that standardizes a subset of CSTA for voice browser based IVR applications, such as those developed in VoiceXML [3] or SALT [4,5]. As speech applications have increasingly grown beyond IVR, though, it becomes evident that making speech an integrated part of the CSTA standards is beneficial and necessary to both speech and telecommunication communities. The new edition of CSTA has taken a first step towards that direction.

In this paper, we review the standards with an emphasis on the provisions related to the spoken language processing. In Sec. 2, we first give an overview on the building blocks of CSTA. We show the basic objects and the services it provides to model the telecommunication infrastructure. In Sec. 3, we show how CSTA objects are constructed to meet the needs of modern spoken language technologies. Finally in Sec. 4 some sample example programs are given.

2. A BRIEF OVERVIEW OF CSTA

CSTA is a standard that specifies programmatic access and control of the telecommunication infrastructure. Software can be developed for a wide variety of communication tasks, ranging from initiating and receiving simple telephone calls to managing large scale multi-site collaborations via voice and video.

2.1. CSTA family

CSTA is standardized in a number of Ecma/ISO specifications whose relationship is shown in Fig. 1. The core operation model and the semantics of the CSTA objects, services and events are defined in ECMA-269. These CSTA features are defined in an abstract and platform independent way so that they can be adapted to various programming environments. In addition, CSTA is accompanied with several standardized programming syntax, among them, ECMA-323 that defines the extensible markup language (XML) binding to CSTA commonly known as CSTA-XML, and ECMA-348, the Web Service Description Language (WSDL) binding. These language bindings, considered as part of the CSTA standard suite (blue boxes in Fig. 1), insure maximum interoperability, i.e., making CSTA features available to computers running different operating systems through any standard transport protocols, including Transmission Control Protocol (TCP), Session Initiation Protocol (SIP), or Simple Object Access Protocol (SOAP).

Thanks to the abstract nature of the core CSTA specification, further extending CSTA to other programming environments is straightforward (illustrated as yellow boxes in Fig. 1). For example, one can apply the design principles of ECMA-335, or Common Language Infrastructure (CLI) [1],

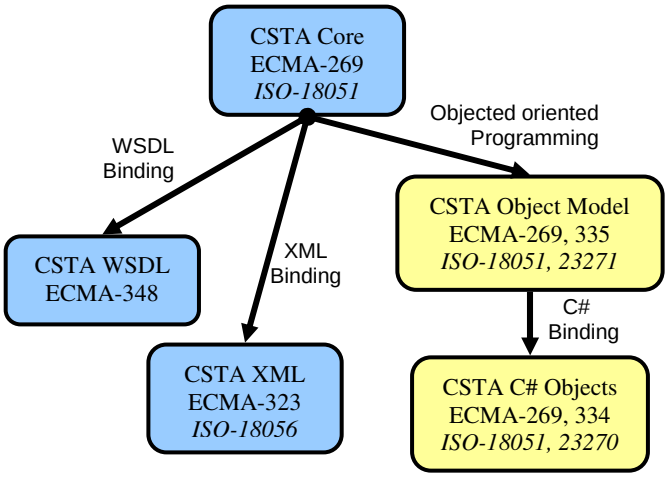

Figure 1: CSTA Standard Suite (blue boxes) and Extensions

and adapt CSTA to an objected oriented programming environment. A key feature of the CLI design is to enable software objects created in one language to be used natively and seamlessly in another language. As a result, software designers can program against CLI-based CSTA object model in any modern programming languages such as ECMAScript, C++ (ISO-14882), and C# (ECMA-334) [1], all covered by international standards, or popular but proprietary languages such as Visual Basic and Java.

Note that many of the mature Ecma specifications eventually become ISO standards sanctioned by national governments. For instance, ECMA-269, 323, 334, and 335 are also known as ISO-18051, 18056, 23270, and 23271, respectively, as shown in Fig. 1.

2.2. Basic concepts

To meet today's global and enterprise communication needs, the functionality requirements on the richness and flexibility of the underlying infrastructure are demanding. CSTA is able to provide comprehensive features for a variety of environments yet with manageable complexity by keeping the basic objects simple.

Under the CSTA model, there are only three objects in a communication system. A *Device* is a logical or physical entity that models the terminal point of a communication link. A CSTA device can receive or initiate communications. A telephone, a fax machine, an automatic call distributor, a predictive dialer, or a voice mail box are all examples of CSTA devices. In the new edition, speech recognizers, natural language parsers and speech synthesizers are introduced as a new class of devices for automatic speech services (Sec. 3).

For historical reasons, a communication session is referred to as a CSTA *Call*. A call object is where session data such as billing information are stored. In many scenarios, it is often desirable that some user data can be moved along with a call so that customers do not have to repeat their personal information every time they are transferred to a new agent, for example. The CSTA call object is where the correlate data can be stored.

Each device in the session is joined to the call by a CSTA *Connection*. This object captures the state and the media characteristics of the relationship between the device and the call. Possible connection states include null, alerting, connected, or held, and the media characteristics can contain information such as voice or multimedia codec, directions of media stream such as duplex, one way or none (muted). As a result of keeping the connection states, the CSTA connection object is where most of the call control services (e.g., deflecting, conferencing or holding calls) are provided. Clearly, the connections for spoken language devices are mostly one way and speech only.

The software keeps track of the status of the system by taking snapshots or placing monitors on the calls or devices to receive event notifications. Fig. 2 illustrates the CSTA model of a user calling into an automatic speech service system on device D1. As the user initiates the call with device D2, the corresponding call C1 and connections D1C1 and D2C1 are created. The connection state for D2C1 is "connected," while the state for the recipient goes from alerting to connected after the automatic agent instructs D1 to accept the call. During the process, the monitor on D1 first raises a *Delivered* event that notifies the software agent of an incoming call. As the connection state transitions into "connected," the *Established* event is raised to indicate that the media channel has been established.

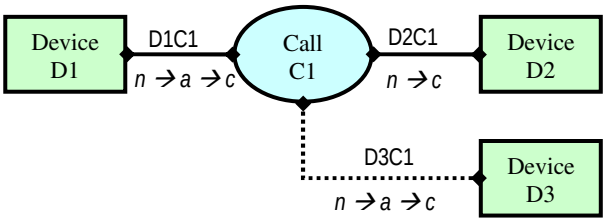

Figure 2: CSTA model for three way communication where a caller uses D2 to call D1 and device D3 is later joined into the call. Connection states null, alerting, and connected are annotated with their initials, respectively.

When the software agent needs speech services, spoken language devices can be dynamically included into the call. To add a speech synthesizer, shown as device D3 in Fig. 2 for example, a conference service can be requested against C1. As a result, a new connection D3C1 is formed, with its connection state transitioning from "alerting" to "connected" as is the case for D1C1. The software agent can place a monitor on D3 or C1 to receive the Delivered and Established events.

3. SPOKEN LANGUAGE SERVICES IN CSTA

The new edition of CSTA accommodates spoken language technologies by introducing a new class of devices specialized for spoken language processing. The new devices are all modeled after a parameterized pattern recognizer that links a CSTA connection to a text channel, as shown in Fig. 3. Generally speaking, all the speech devices can be viewed as performing a maximum *a posteriori* (MAP) decision

$$z = \arg\max_{y \in \Omega} P(y \mid x) \qquad (1)$$

This model applies to devices that process audio input or generate audio outputs. The CSTA speech input device is called a *Listener*, while the output, a *Prompt*. A Listener device can be configured for speech recognition/understanding, and speaker identification/verification. Similarly, a Prompt device can be used to synthesize natural language sentences in speech, optionally mixed with other pre-recorded audio segments.

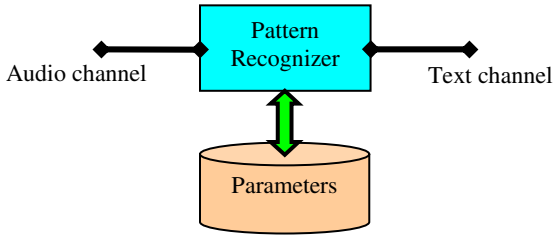

Figure 3: Schematic of CSTA spoken language devices

3.1. Listener

The Listener devices use the CSTA connection as the audio source x, and produces the textual outcome z based on the parameters $P(y \mid x)$ and search space Ω. The model described in (1) is designed to be flexible and extensible for accommodating current and future technology improvements. For example, to configure the CSTA device for speech recognition or understanding, one can include the semantic language model and the semantic schema as the parameters and the search space [6,7], respectively. The textual outcome may be in the format of semantic markup language (SML) [6,8] or the Extensible Multimodal Annotation markup language [9]. When ready, these documents are reported back to the application using the standard CSTA event mechanism.

For speaker verification, the parameters and the search space can be the speaker and the cohort models, respectively. As shown in Fig. 3, the input device may update its parameters based on the audio sessions it has been processed, using MLLR [10] or MAP [11] adaptation algorithms as appropriate. In other words, a Listener device can be configured for speaker dependent or speaker adapted speech recognition, as well as for the enrollment for speaker identification or verification.

The rich call controls provided by CSTA allow multiple devices to be joined to the same communication session. It is therefore possible to perform multiple simultaneous speech input processing. For example, one can configure a Listener device for speech recognition and another for speaker verification, and use CSTA Join Call service to combine them together in a conference call setting. The audio is forked to both Listener devices, which then report their individual outcomes by raising respective CSTA events.

Similarly, it is straightforward to share a Listener device among multiple sessions. This is especially useful when the Listener is used to perform a resource demanding task, such as recognizing proper names from a big database. The mechanism of sharing is well defined in CSTA: every time a device is added to a conference call, a new connection is created to uniquely identify the audio fork from the call to the device. All the consequent commands and events, such as Start that kicks off the recognition and the Recognized event raised when the result is ready, are all connection based. In other words, the CSTA specification makes it very clear that which audio should be taken as the source for input, and which session the outcome is reporting to.

3.2. Prompt

In contrast, the Prompt devices use the CSTA connection as the audio destination z, and converts the textual input x into audio waveform. Although (1) seems to imply the text to speech conversion must adhere to certain statistical waveform concatenation based algorithm (e.g., [12,13]), it is believed that model-based approaches can be appropriately parameterized and incorporated into a CSTA device as well. On the other hand, the majority of the synthesizers deployed by the industry do use waveform concatenation techniques. There, the search space Ω includes the waveform segments to be pieced together, while the parameters $P(y \mid x)$ is a model that measures the match of the given text to the potential waveform segments based on prosodic fitness. The text input format currently recognized by the Prompt device is W3C Speech Synthesis Markup Language [14] with an optional extension from SALT.

Unlike speech input where the outcomes must be obtained by capturing CSTA events, it is possible to use a Prompt device without listening to any of its events. To facilitate rich user experience, however, CSTA defines a set of events that allow program designers to fine tune application behaviors based on the playback progress. For example, the Prompt device raises *Barge In Detected* event if certain user input activities are detected during the prompt playback. The application can then inspect the nature of the interruption and take appropriate actions. If the input is a dialog act commencing a user turn, the prompt playback may be terminated. However, if the user input is simply some back channel communication, such as a request to adjust the playback speed or volume, the playback may continue with proper adjustments. A designer can also insert into SSML named bookmarks so that the Prompt device can raise a *Bookmark Reached* event when the text prior to the bookmark has been rendered. This event is useful to determine whether a barge-in activity bears any semantic implications under the dialog context, or can be simply used to synchronize lip shape or other screen display in a multimedia presentation.

Audio playback is a sequential process in nature. As a result, CSTA provides another device, called *Prompt Queue*, to coordinate the playback. Prompt Queue shares and integrates well with the existing CSTA voice unit device that models a voice mail management system. Multiple synthesized segments can be queued up in Prompt Queue, for which CSTA provides rich controls to dynamically adjust the speed and volume, suspend and resume, reposition or discard individual prompts.

4. ILLUSTRATIVE EXAMPLES

The design principle of CSTA underlines a fundamental belief that complicated and sophisticated systems can be approached with a modest amount of simple but well defined objects. Despite the features demanded for modern communication systems, CSTA utilizes only three objects to describe the infrastructure underlying them. Following this principle, the spoken language technologies are introduced with a simple object implementing (1), which happens to be the core of the design of SALT for spoken language processing as well [5]. As

a result, the spoken language processing portion of a CSTA program bears close resemblance to that written in SALT, and vice versa.

Although SALT is designed for programming in a markup language environment, the underlying objects can be abstracted and applied to programming languages in the CLI family targeted by CSTA. For example, a typical program to set up a speech recognition in SALT may appear as follows:

```
<listen id="myReco"
     onreco="h1()"
     onnoreco="h2()">
  <param name="server">sip:myasr</param>
  <grammar name="main" src="http:..."/>
</listen>
```

The program instantiates a speech input object `myReco` with a grammar `main` through a URI reference, using the acoustic model and the search engine located at `sip:myasr`. Event handlers `h1` and `h2` will be invoked when the speech is recognized or rejected, respectively. Under CSTA, the equivalent C# program is

```
myReco = new Listener();
myReco.Recognized +=
     new RecognizedEventHandler(h1);
myReco.NotRecognized +=
     new NotRecognizedEventHandlre(h2);
myReco.Server = new Uri("sip:...");
myReco.Grammars.Add("main",
     new UriGrammar("http:..."));
```

Other than some name changes, the two programs are virtually identical. The naming of CSTA objects and events follows the convention specified in ECMA-335, where nouns and past participles are used for object and event names. In contrast, SALT follows mostly the HTML convention. As a result, the `listen` tag in SALT maps to the `Listener` object in CSTA whereas `onreco` and `onnoreco` events in SALT, to the `Recognized` and `NotRecognized` in CSTA. In either case, the application starts the recognition using the command

```
myReco.Start();
```

Both SALT and CSTA obtain the results via the event handlers. For example, under HTML and ECMAScript, the function that handles a successful recognition event for the above SALT example may appear as follows:

```
function h1()
{
     res = event.srcElement.recoresult;
     n = res.selectSingleNode("//names");
     ...
}
```

The source object that raises the event is accessible through the global HTML object `event.srcElement`, and the recognition result, the `recoresult` property. The same program in C# is as follows:

```
void h1(object s, RecognizedEventArgs args)
{
     Result  res = args.result;
     XmlNode n = res.SelectSingleNode("..");
     ...
}
```

where the event source is passed on to the event handler as an argument `args`. Other than the slightly different syntax originated from the manners in which HTML and CLI handle events, the two programs are isomorphic. In both cases, the SML can be queried against via the `res` object using standard, programming language independent methods for XML, such as the `SelectSingleNode` shown above.

5. SUMMARY

This article describes the international ECMA/ISO standard for speech application program interface. The open standard ushers in an era where speech applications can be created in a portable and interoperable fashion as speech applications are becoming prevalent.

The standard is based on SALT. Given the success of the SALT deployments for spoken language applications, it is reasonable to believe that the similarity to SALT will see the spoken language provisions in this standard provide designers with rich enough features to meet the needs of modern speech application.

6. REFERENCES

[1] Ecma International, http://www.ecma-international.org.

[2] World Wide Web Consortium (W3C), Hypertext Markup Language Specification, version 4.01, December 1999.

[3] W3C, VoiceXML Specification, version 2.0, March 2004.

[4] SALT Forum, Speech Application Language Tags Specification, version 1.0, June 2002.

[5] K. Wang, "SALT: Spoken Language Interface for Web-based Multimodal Dialog Systems," in *Proc. ICSLP-2002*, Denver CO, October 2002.

[6] K. Wang, "Implementation of a multimodal dialog system using extensible markup language," in *Proc. ICSLP-2000*, Beijing China, October 2000.

[7] K. Wang, "A plan based dialog system with probabilistic inferences," in *Proc. ICSLP-2000*, Beijing China, October 2000.

[8] K. Wang, Y. Wang, and A. Acero, "The use and acquisition of semantic language model," in *Proc. NAACL/HLT-2004*, Boston MA, May 2004.

[9] W3C, Extensible Multimodal Annotation (EMMA) Markup Language, working draft, March 2004.

[10] C. Leggetter and P. Woodland, "Flexible speaker adaptation using maximum likelihood linear regression," in *Proc. EuroSpeech-95*, Madrid Spain, August 1995.

[11] J.-L Gauvain and C.-H Lee, "Maximum *a posteriori* estimation for multivariate Gaussian mixture observations of Markov chains," *IEEE Trans. Speech and Audio Processing*, Vol. 2, No. 2, pp. 291-298, April 1994.

[12] X. Huang *et al*., "Whistler: A trainable text to speech system," in *Proc. ICSLP-96*, Philadelphia PA, October 1996.

[13] A. Acero, "Formant analysis and synthesis using hidden Markov models," in Proc. EuroSpeech-99, Budapest Hungary, September 1999.

[14] W3C, Speech Synthesis Markup Language, candidate recommendation, December 2003.