A Speech Mashup Framework for Multimodal Mobile Services

Giuseppe Di Fabbrizio
AT&T Labs - Research
180 Park Avenue
Florham Park, NJ 07932 - USA
pino@research.att.com

Thomas Okken
AT&T Labs - Research, Inc.
180 Park Avenue
Florham Park, NJ 07932 - USA
tokken@research.att.com

Jay G. Wilpon
AT&T Labs - Research, Inc.
180 Park Avenue
Florham Park, NJ 07932 - USA
jgw@research.att.com

ABSTRACT

Amid today's proliferation of Web content and mobile phones with broadband data access, interacting with small-form factor devices is still cumbersome. Spoken interaction could overcome the input limitations of mobile devices, but running an automatic speech recognizer with the limited computational capabilities of a mobile device becomes an impossible challenge when large vocabularies for speech recognition must often be updated with dynamic content. One popular option is to move the speech processing resources into the network by concentrating the heavy computation load onto server farms. Although successful services have exploited this approach, it is unclear how such a model can be generalized to a large range of mobile applications and how to scale it for large deployments. To address these challenges we introduce the *AT&T speech mashup* architecture, a novel approach to speech services that leverages web services and cloud computing to make it easier to combine web content and speech processing. We show that this new compositional method is suitable for integrating automatic speech recognition and text-to-speech synthesis resources into real multimodal mobile services. The generality of this method allows researchers and speech practitioners to explore a countless variety of mobile multimodal services with a finer grain of control and richer multimedia interfaces. Moreover, we demonstrate that the speech mashup is scalable and particularly optimized to minimize round trips in the mobile network, reducing latency for better user experience.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Web-based services; H.5.3 [**Group and Organization Interfaces**]: Web-based interaction; H.5 [**Information Interfaces and Presentation**]; D.2.11 [**Software Architectures**]

General Terms

Experimentation, design

Keywords

Mashups, speech mashup, speech services, speech system architecture, web services, multimodal

ICMI-MLMI'09, November 2–4, 2009, Cambridge, MA, USA.

1. INTRODUCTION

Accessing information and services over the web is a daily routine for professionals and casual web surfers. Users track parcels in real time, find a nearby business, or check a restaurant review, all by typing a few keystrokes on a desktop and without paying much attention to the rather complex web services orchestration hidden behind the curtains. Thanks to the proliferation of service-oriented architecture (SOA) [21] and public web-based interfaces, producing new web services has become much easier, within reach of ordinary developers. Major web industry players are now opening up the "walled garden" of proprietary content to allow consumers to access their technology to play with maps, track order information, receive real-time news feeds, and create new services simply by combining existing web services.

Recently published web service interfaces such as Google Maps, Yahoo! Flickr, Amazon Web Services, and YELLOWPAGES.COM™, greatly simplified the creation of third-party applications by hiding the complexity of the technology into the network. A classic example is the use of geographic information from Google Maps[1] and real estate data from CRAIGLIST[2] to generate a new distinct service where houses for rent or sale are visualized on an interactive map of the USA[3]. The process of merging several data sources in a single integrated application has been defined as *mashup* or *web application hybrid* [27] and has become a popular method for combining public application programming interfaces (API) and RSS[4] data feeds.

Conversely, traditional speech-enabled services rely on a well-established telephony programming model and architecture. Media resources, such as automatic speech recognition (ASR) engines, text-to-speech (TTS) synthesis engines, audio players, recorders, and touch-tone detectors, are tightly bound together with the telephony interface by an event-driven media resource manager [12]. Typically, an orthogonal control layer manages the fine-grained details of media synchronization, resource allocation, audio streaming routing, and telephony signaling. VoiceXML-based systems [24], for instance, hide these mechanical details by exposing a higher-level API that handles most of the resource management *minutiae* [12].

However, developers are still required to understand the underlying reactive nature of the media resource interaction [6]. For example, prompts in VoiceXML are queued and played only when the execution reaches an input state implicitly defined in the form execution. The semantics of VoiceXML markup interpretation hinges on the controversial Form Interpretation Algorithm (FIA), which

[1] maps.google.com
[2] www.craiglist.org
[3] www.housingmaps.com
[4] Really Simple Syndication data format.

spans across the combined resources execution and implicitly defines a complex state machine execution. Other approaches to speech services such as XHTML+Voice (X+V) [4] and the neglected Microsoft's Speech Application Language Tags (SALT) [26] introduce multimodality into the equation so that, in addition to speech, input and output can be conveyed through a graphical user interface usually hosted by a web browser. Input modalities are integrated by explicitly synchronizing speech grammars and text form inputs.

In this sense, speech services design depends upon a monolithic architecture and closely coupled components that make it difficult to use the speech components in a context different from the traditional telephony execution environment. In the web application domain, the concept of loosely coupled and interoperable components, as found in many SOAs, has been a successful programming paradigm for reliable and scalable service composition. Distributed resources are made available as independent services through simple communications protocols and API publication mechanisms (SOAP, WSDL, UUDI, etc. see [3, Chapter 6] for a detailed description).

These considerations suggest that techniques developed in the area of web services could be beneficial to speech service when the data channel is used as a communication link. More specifically, can speech processing be seamlessly integrated into a web environment to leverage such broadly accepted programming paradigms and facilitate the creation of new mobile multimodal services? Can speech processing resources be decoupled and made stateless building blocks available through the network with simple APIs? The *AT&T speech mashup* architecture addresses these challenges by integrating speech and web services into one consistent network-hosted application framework for multimedia devices with broadband access (iPhone, BlackBerry®, IPTV set-top box, SmartPhones, etc.) without the need to install, configure, or manage speech processing software and equipment. Speech mashups allow researchers and speech practitioners to easily and rapidly develop new speech and multimodal mobile services as well as new web-based services by combining speech processing and web services similarly to the web mashup approach.

The rest of the paper is organized as follows. Section 2 describes the overall AT&T speech mashup architecture. Section 3 pinpoints issues related to the use of data streaming over the wireless network, the network APIs, and the supported data formats. Section 4 shows the possible architecture configurations, while 5 describes the web portal used for speech resource management. Section 6 illustrates the mobile clients currently supported, and section 7 shows some sample applications implemented with this framework. Some preliminary evaluation of the network latency is reported in section 8. Finally, section 9 compares the speech mashup framework with existing similar architectures, and section 10 summarizes the paper.

2. ARCHITECTURE

The AT&T speech mashup is a web service that implements speech technologies, including both automatic speech recognition and text-to-speech synthesis, for web applications. This enables users of an application to use voice commands to make requests or to convert text to audio. Speech mashups work by relaying audio or text from an application running on a mobile device or a web browser to servers at the AT&T network where the appropriate conversion takes place. The result of the process, either text or audio, is returned to the client application. Speech mashups can be created for almost any mobile device, including the iPhone, as well as web browsers running on a PC or Mac, or any network-enabled device with audio input/output capabilities.

The concept behind the speech mashup technology is intuitive and similar to the familiar web application approach. Communication between devices and the service platform is established over the packet-switched network; no traditional circuit switching sessions take place. As pictured in Figure 1, in a typical speech recognition interaction, the speech is first captured on the mobile device (*the client*) through the microphone and compressed using one of the available speech codecs (for example the Adaptive Multi-Rate[5] codec at 12.2 kbits/s). Then an HTTP (Hypertext Transfer Protocol) [7] connection (*the transport*) is established with the speech mashup manager (*the server*), which delivers the bit stream to the AT&T WATSON speech recognizer engine [16] along with a set of parameters including the reference to the grammar or language model used to recognize the utterance. The recognition results are then returned back to the client and used by the client to take the next action. Depending on the complexity of the task, a semantic interpretation could be added to the results so that natural language variation of the same user's intent can be interpreted properly. Optionally, an application or web mashup server could provide additional service logic that is not strictly related to speech processing.

The speech mashup manager (SMM) layer implements a multi-user, multi-application hosting platform with two main functions: 1) run-time resource allocation management, 2) off-line user account management. In the former case, it makes the AT&T WATSON speech recognition engine and the AT&T Natural Voices text-to-speech synthesis engine accessible through any network as web services and exposes them through an HTTP-based API. In the latter function, the SMM provides web interfaces to upload and compile users' grammars, data loggers to trace the service activities, and tools for utterance transcription.

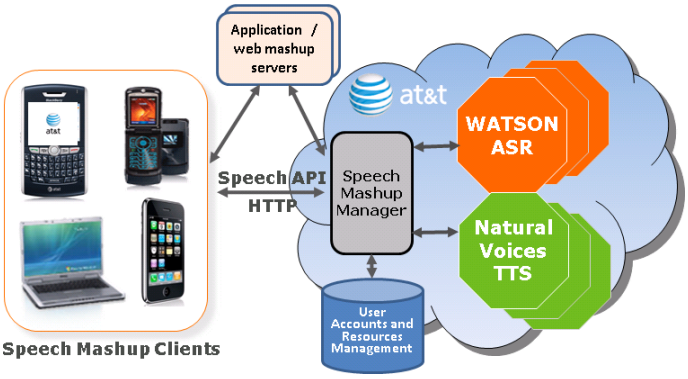

Figure 1: AT&T speech mashup architecture

From the architecture perspective, a speech mashup application consists of three main components:

1. the AT&T speech mashup server (SMS), including the speech mashup manager (SMM), the WATSON servers for ASR, and the Natural Voices servers for TTS. The SMM opens and manages direct connections to the appropriate AT&T speech servers on behalf of the client, including resolving device-dependency issues and performing authentication and general accounting.

2. A speech mashup client that relays audio to the AT&T speech mashup servers and accepts the recognition result and/or a client that receives audio buffers and plays them through the

[5] en.wikipedia.org/wiki/Adaptive_multi-rate_compression

audio interface. Examples of speech mashup clients are available for Java ME[6] devices, the iPhone, and the Safari browser on the Mac OS X.

3. A main application server (e.g., Apache or Tomcat) or a web mashup server (e.g., WSO_2 Mashup Server[7]) that provides access to the application's back-end database, performs data aggregation processing with other application servers, and, depending on the adopted mashup configuration, could implement the application logic.

The SMM component is implemented as a Java servlet with a relational database back-end, while the other components are native Linux processes. No special hardware is required; all processes run on Linux servers and have been tested in the Amazon Elastic Compute Cloud (Amazon EC2[8]) for large-scale deployments.

3. NETWORK TRANSPORT

As broadband access becomes available for mobile devices, delivering speech over data channels is a natural choice for speech applications. However, streaming of continuous media over wireless links is a difficult problem due to the real-time nature of the media generated by wireless clients.

3.1 Speech over the data network

A variety of protocols are available for real-time media transmission over IP networks. The Media Resource Control Protocol version 2 (MRCPv2) [25], for example, is an IETF recommendation for speech servers communication that relies on Real Time Streaming Protocol (RTSP) and Session Initiation Protocol (SIP). While MRCPcv2 has been designed to deliver real-time media over IP, it was not expressly designed for the wireless network. Moreover, the implementation complexity of both client and server side might be overkill for mobile applications.

The wireless 3rd generation (3G) network specifies a maximum data rate of 2.4 Mbits/s on the download link and 153.6 kbits/s on the upload link shared by all users within a single cell sector. These are usually ideal bandwidth conditions that tend to degrade due to environmental conditions such as fading and interferences or other conditions like the device travel speed and the density of devices per cell. To deliver media over wireless networks, it is important that mobile clients be carefully designed to reduce latency by compressing media and reducing the number of retransmissions.

Based on the previous considerations, HTTP [7] has been selected as the transport protocol for the speech mashup framework. HTTP has been originally designed to support HTML, but with the recent version 1.1 extensions, it has been often used as main choice for media protocol on the web. HTTP 1.1 supports persistent connections, allowing sockets to be reused over lengthy transmissions, eliminating the overhead of creating new connections for every transaction. HTTP 1.1 also implements chunked encoding, which permits HTTP messages to be broken into several parts. This is very important for delivering and receiving speech segments as they are produced or consumed by the wireless client. Although real-time streaming is not guaranteed, message chunking and persistent connections greatly reduce latency, achieving real-time performance in the majority of working conditions (see section 8 for latency evaluation). In essence, HTTP, although not strictly designed for real-time media streaming, turn out to be a practical trade-off between simplicity and efficiency.

[6]Java Micro Edition - `java.sun.com/javame`

[7]`wso2.org/projects/mashup`

[8]`aws.amazon.com/ec2/`

Table 1: ASR REST API (partial list)

Parameter	Value	Description
uuid	string	Required. Unique user ID assigned at registration.
resultFormat	string	Optional. Result format, which can be EMMA, JSON, XML.
appname	string	Required. Name of application directory.
cmd	string	Required. One of the following command strings:
		oneshot, rawoneshot
		Starts ASR in stateless mode; the request body will contain the entire audio stream.
		start Starts ASR in stateful mode; the audio stream will be sent using one or more audio or rawaudio requests.
		stop Stops stateful ASR and returns the result.
		audio, rawaudio, Sends a chunk of audio for stateful ASR
audioFormat	String	Optional. This specifies the format of the audio data supplied by the client. Possible values are:
		amr Adaptive Multi-Rate (AMR), narrow-band only
		au Sun AU, μ-law or 16-bit linear
		caf Apple Core Audio Format, mu-law or linear
		wav Microsoft/IBM Wave, μ-law or linear
		...

3.2 REST-based APIs

The speech mashup API follows the Representational State Transfer (REST) [15] network architecture principles. REST refers to resources in the network as uniquely identified by a global identifier (e.g., a URI or Uniform Resource Identifier on HTTP). For example `http://service.research.att.com/smm/asr?uuid=[uuid]&grammar=citystate&format=emma` is a request directed to the speech recognizer resource (`/smm/asr`) with the grammar `citystate` by the registered user identified by the unique user ID (`uuid`). Requests for resource manipulation are sent to components in the network that are responsible for responding to the request with a *representation* of the request, for instance an XML document containing the result of the speech recognition process following the Extensible MultiModal Annotation (EMMA) markup language [18] standard and declared as `format=emma` in the previous example. In the case of a speech recognition interaction, a RESTful call is carried out by the client over an HTTP POST request where the API parameters are in the HTTP header and the speech data attached to the HTTP body. Multiple chunks of data are fleshed out in a sequence of HTTP POSTs by the client as the speech is captured by the microphone. When the utterance is completed, the server generates a response in the specified output format. Table 1 summarizes the main APIs for the ASR.

In the simplest form, a client request can be carried out by the traditional Unix `wget` command (Figure 2) where the input speech is stored in a file (`sample-speech.amr`) and the result written as EMMA document in a text file (`response.emma`).

```
wget \
  --post-file=sample-speech.amr \
  --header 'Content-Type: audio/amr' \
  --server-response 'http://service.research.att.com/smm/asr?  \
                     grammar=citystate&uuid=[uuid]&result=emma' \
  -O response.emma
```

Figure 2: `wget` speech recognition client request

Conversely, a TTS interaction starts by posting the text to synthesize and the server responds with chunked data stream over a persistent data connection. TTS bookmarks can be interleaved with the speech chunks as well. The client has the responsibility for playing the speech with the proper sample rate and for dispatching the bookmarks at the proper time. Table 2 shows the REST API for the TTS resource.

Table 2: TTS REST API (partial list)

Parameter	Value	Description
uuid	string	Required. Unique user ID assigned at registration.
text	string	Optional. Text may also be supplied in the body of a POST request.
audioFormat	string	Optional. This specifies the format of the audio data supplied by the client. Possible values are:
		amr Adaptive multi-rate (AMR), narrow-band only
		mulaw AU with mu-law encoding
		alaw AU with A-law encoding
		linear AU, 16-bit linear
voice	string	Optional. Crystal (default) or Mike.
sampleRate	integer	Optional. The audio data sample rate. Defaults to 8000 Hz. Note that AMR-NB has a fixed sample rate of 8000 Hz, so specifying a different sampleRate will produce odd-sounding results.
ssml	True or False	Optional. Set this parameter to True when text contains SSML tags. (When set to the default, False, each word is pronounced, including SSML tags).
		...

3.3 Supported data formats

The SMM returns or accepts structured data related to the execution of a specific speech processing task to or from the client. For an ASR task, speech recognition results and detailed information about the parameter values and settings used during the recognition are described in different equivalent formats that can be selected based on the client capabilities or preferences. There are three standard output formats: XML, JavaScript Object Notation (JSON) [11], and EMMA markup language [18]. JSON is a data serialization format that maps directly into JavaScript objects and it is particularly convenient in a browser-based client environment where a JavaScript interpreter is natively available. EMMA is a richer notation recommended by W3C for interoperable input format representation for multimodal systems. EMMA facilitates plug-and-play of systems components and is suitable for annotating various stages of the processing of the user's input. An example of EMMA document is shown in Figure 3. In this case, the ASR recognized the utterance *"a large pizza with pepperoni sausage two Diet Pepsi and a Root Beer"* which is captured in the element `emma:interpretation` both as literal text (i.e., attribute `emma:token`) and semantically tagged interpretation (i.e., body of the element `interpretation` with semantic XML markups).

For a TTS task, the input text follows the W3C Speech Synthesis Markup Language (SSML) [9] standard. SSML is an XML markup language for modifying the way text is processed by TTS engines. The SSML tags are instructions for normalizing text and controlling emphasis and other speaking qualities (prosody). For example, the attribute `type = telephone` treats the text as a telephone number so that the SSML fragment `<say-as type = "telephone"> 9081234567 </say-as>` is synthesized as *"nine zero eight [pause] one two three [pause] four five six eight"*.

4. SPEECH MASHUP CONFIGURATIONS

Speech mashups can be configured in different ways depending on the application complexity, the wireless network latency, and the level of client authentication required. Figure 4 shows a typical **client-side** configuration where the N clients $(C_1, ..., C_N)$ implement the logic of the application and are responsible for the communication between the speech mashup server (SMS) and the application / web mashup servers (AS). All the network connections are originated by the mobile clients in the wireless packet-switched

```
<emma:emma version="1.0">
<emma:grammar id="gram1"  ref="smm:grammar=ipizza&amp;UUID=[uuid]"/>
<emma:model id="model1"  ref="smm:file=pizzahut.xsd&amp;UUID=[uuid]"/>
<emma:info>
    <session_id>33C07738-DA61-4814-B60D-4D374F143D8D</session_id>
</emma:info>
<emma:one-of id="one-of1" emma:medium="acoustic"
  emma:mode="voice"  emma:function="dialog"
  emma:verbal="true"  emma:lang="en-US"
  emma:start="1246437000"  emma:end="1246437008"
  emma:grammar-ref="gram1"
  emma:signal="smm:UUID=[uuid]&amp;file=audio-454907.amr"
  emma:signal-size="8070"
  emma:media-type="audio/amr; rate=8000"
  emma:source="smm:platform=null&amp;device_id=null"
  emma:process="smm:type=asr&amp;version=watson-6.3.0000"
  emma:duration="6150"
  emma:model-ref="model1"
  emma:dialog-turn="33C07738-DA61-4814-B60D-4D374F143D8D:1">
<emma:interpretation id="nbest1"  emma:confidence="1.0"
  emma:tokens="a large pizza with pepperoni sausage two
Diet Pepsi and a Root Beer">
  <item>a
   <size_LG>large</size_LG>
   <type_own>pizza with
   <topm_MPE>pepperoni  </topm_MPE>
   <topm_MIS>sausage</topm_MIS>
   </type_own>
  </item>
  <item>
   <qt_2>two</qt_2>
   <drinks_DP>Diet Pepsi</drinks_DP>
  </item>and
  <item>
   <qt_1>a</qt_1>
   <drinks_RB>Root Beer</drinks_RB>
  </item>
</emma:interpretation>
</emma:one-of>
</emma:emma>
```

Figure 3: Sample EMMA document

network (typically a GPRS[9] or an EDGE[10] network). Speech data requests (dotted arrows) are directed to the SMS component and text data requests (solid arrows), such as database queries, are sent to the AS modules. Each service is identified by a URL and, as described in section 3.2, HTTP is the communication link between the client and the speech mashup server.

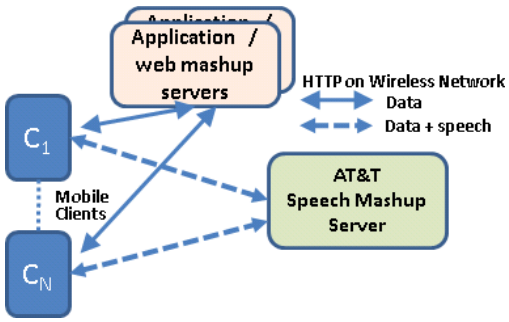

Figure 4: Client-side speech mashup

If the wireless network is congested, creating an HTTP link could take a significant amount of time, limiting the use of this configuration to simpler applications where the number of client round trips is small or a faster wireless network (WiFi) is available. Also, device authentication, when required, is delegated to the SMM component which may involve some application-dependent authentication model directly implemented in the speech mashup server.

[9]General Packet Radio Service
[10]Enhanced Data Rates for GSM Evolution

Figure 4 illustrates a **server-side** configuration. Here the mobile devices are connecting directly to the main application server where the content aggregation with other content servers takes place. In this case, the clients are facing one server with a single URL allowing the application to reuse the connections for multiple interactions, saving on the cost of creating multiple connections. The main application server can use the faster IP network to aggregate the data, format the output for specific device display capabilities, and saving further round trips and processing time on the device. The disadvantage in this configuration is that the speech chunks delivered over HTTP by the client (or produced by the TTS from SMS) have to be relayed to the SMS component (or transferred from the SMS to the client) through the application server. The application server has to implement the HTTP chunking streaming module to pass the speech to the SMS (or vice versa for the TTS) in a timely manner.

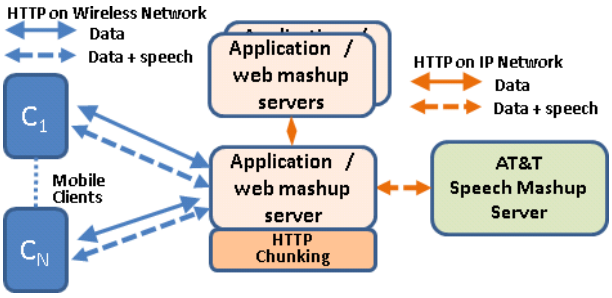

Figure 5: Server-side speech mashup

Finally, Figure 6 sketches the **back-to-back** configuration. As previously mentioned, multiple handset client connections (i.e., HTTP sessions) are significantly more expensive than one connection to accomplish a given task. Passing a large amount of data from a server to a handset for the purpose of forwarding that data to another server (server-side configuration) is far more expensive than having an intermediate server act as a data broker. To address these concerns, SMS allows a single incoming HTTP POST request from a client to specify a preprocessing URL, a speech action (such as a recognition), and a postprocessing URL. The preprocessing and postprocessing URLs are optional. When the SMS receives a request, it first makes an HTTP POST request to the preprocessing URL, passing the body of incoming request as the body of the POST.

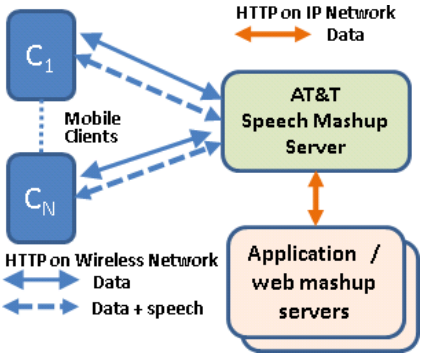

Figure 6: Back-to-back speech mashup

SMM performs the speech action using the response from the preprocessing URL. Then it makes a POST request to the postprocessing URL, passing the result of the speech action as the body of the POST. Finally, it returns the response from this POST to the client as the HTTP response to the client's initial request. This is equivalent to *piping* requests from the wireless network to the IP network and minimizes the number of roundtrips over the wireless network.

4.1 Same origin security policy limitations

Web browser-based access to data implements a sand-box mechanism called *same origin policy*, to forbid access to resources different from the originating site. This is an important security feature that avoids access to authentication information (HTTP cookies) by malicious sites through *cross-site scripting* (XSS) attacks [19]. When using a browser-based client, the back-to-back configuration helps to mitigate possible XSS attacks by allowing SMM to act as trusted proxy for cross-domain communication. In this way, only predefined sites can be referred by SMM, avoiding malicious JavaScript code injections from extraneous sites.

5. SPEECH MASHUP PORTAL

In order to centralize the speech processing resources into the network, a web-based portal has been created to manage user's accounts, ASR grammars, disk space quota management, load balancing across ASR and TTS servers, and logging. Typically, once the speech mashup user registers to the portal, a unique account ID is emailed to the user, a grammar compilation area is created and the user can immediately start using the ASR by referring to the default built-in grammars and the TTS by selecting the available voices. The portal supports any language model that can be represented as weighted finite state machine (FSM), and compiles both rule-based and stochastic grammars into FSMs. In general, rule-based grammars are used for lower complexity tasks and when training data is not available or scarce. Examples of rule-based grammars included in the portal are *dates*, *times*, *confirmation*, *phone numbers*, and *US city/states* . Rule-based grammars can be either represented in SRGS (Speech Recognition Grammar Specification) [20] format or in a more compact proprietary representation similar to the Backus Naur Form notation. Corpus-based stochastic language models are used for more complex tasks like customer care call routing or voice search and are compiled into a weighted FSM with Katz back-off smoothing [2] (other smoothing methods are also available). A typical natural language customer care task requires an average of 5,000 transcribed utterance samples to reach operational performances (>75% word accuracy).

Each grammar can be assigned to a different user's account scope:

- **My grammars**. User's grammars that are not accessible to anyone else.
- **Shared grammars**. Grammars created by others and then made available to all speech mashup users.
- **Built-in grammars**. English and Spanish grammars included with the speech mashup including grammars for US cities, digits, account numbers, and US last names.

The portal also exposes logging files for debugging and tracing purposes and a web-based transcription tool to listen and transcribe the recorded utterances in real time. Utterance transcriptions can be periodically downloaded and used as training data to improve the speech recognition performances.

6. MOBILE CLIENTS

A speech mashup client (Figure 8) is the part of a speech-enabled application that runs on the user's mobile device or, in principle,

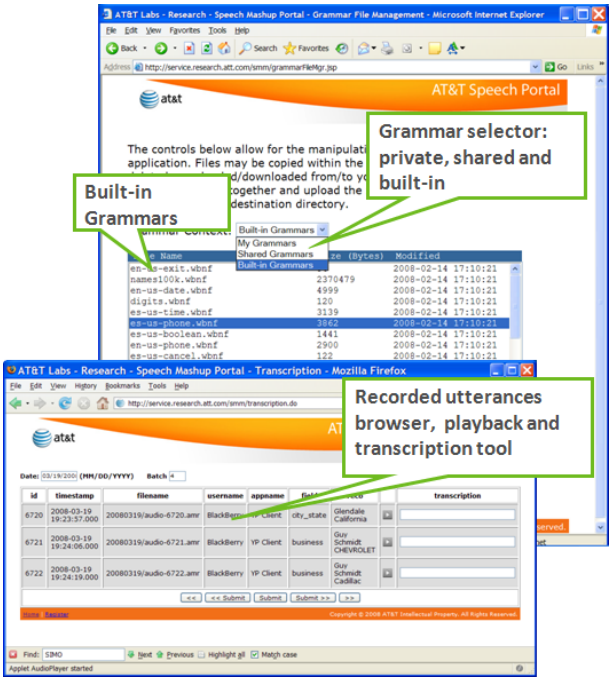

Figure 7: AT&T speech mashup portal grammar manager and utterance transcription screenshots

on any voice-enabled and networked device. Its role is to capture and relay speech or text to the speech mashup manager, which in turn handles authentication, accounting, and communication with the AT&T speech servers. The AT&T speech mashup portal web site provides the following client examples downloadable in source code through the sample code link:

- A Java ME client that can be used for most Java-enabled mobile devices.
- A native client application for the iPhone.
- A web browser (Safari) plug-in for Mac OS. The plug-in streams audio from the desktop microphone to the SMM and implements a JavaScript API integrated with the HTML browser environment.

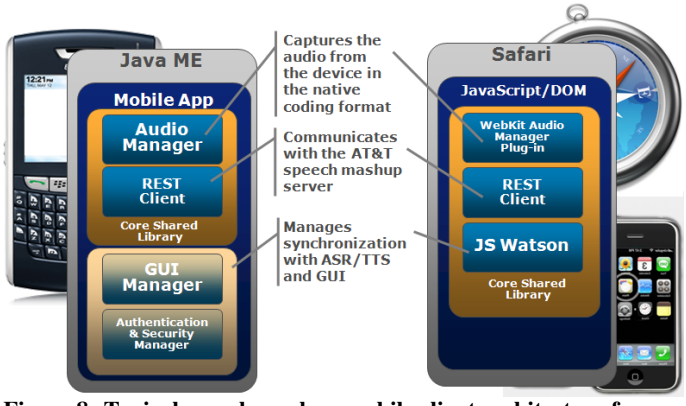

Figure 8: Typical speech mashup mobile client architecture for Java ME and Mac OS browsers

The client code samples include three main modules: 1) an audio manger to capture real-time speech from the device microphone; 2) a REST client that implements HTTP and the speech mashup server API; 3) a simple graphical user interface (GUI manager) that enables a 'Speak' button on the device (either an actual button on the mobile keypad or a touch screen soft button). After the user hits the speak button, the audio capture process starts and the speech samples are streamed over HTTP to the ASR through the speech mashup server. When the button is released, the mashup returns an XML or JSON document containing the speech recognition results which is simply displayed on the device screen. In general, an interaction manager would be responsible for taking the next action to present to the user.

7. SAMPLE APPLICATIONS

This section illustrates three multimodal mobile examples implementing the three architecture configurations described in section 4. Input modalities are either speech or typed text. The first is a native iPhone application for local business search freely available in the Apple Store. The second is a prototype iPhone service for pizza ordering. The last example is a version of mobile local business search for BlackBerry phones.

7.1 speak4it

speak4it[11] [14] is a native iPhone application for local business search for accessing around 20M entries from the YELLOWPAGES.COM search engine. It is based on the **server-side** configuration, where the application server combines the speech requests with a local business search engine and a map location information server. The user inputs a request by holding the Push & Talk button and uttering a natural language query such as *"Find Japanese restaurants in Glendale California"*. Voice search queries can be either by business name or category whereas a natural language understanding module extracts the information needed to populate the query fields for the search engine. If the location is omitted, the iPhone location system information is used as default coordinates. The result of the search is displayed in the form of a listing or as scrollable map. The user can call any item in the list by tapping on the displayed phone numbers.

7.2 iPizza

iPizza (Figure 9 and 10) implements a multimodal interface for online pizza ordering based on the **client-side** configuration. It runs as native application on an iPhone and combines speech and touch inputs with graphical interaction to enable users to rapidly select and edit menu items. Users can speak naturally by tapping on the 'Talk' button and request multiple items at the same time. For example: *"I'd like to order a pizza with mushrooms and ham, two diet Pepsi and baked cinnamon sticks"*. The graphical interface allows easy navigation to update the items in the shopping cart by mixing voice and touch inputs at any stage of the ordering process.

Items listed in the shopping cart (Figure 9) can be edited in details by tapping on a specific item to edit or by saying the new value of a field (Figure 10) by using the talk button.

7.3 JME local business search

The JME local business search is similar to speak4it, but implemented as Java ME application and it runs on the BlackBerry family of smartphones. The user formulates the voice search query by holding down the keypad call button for the duration of the utterance. This example exercises the **back-to-back** configuration where the speech recognition results are first analyzed by the natural language understanding component and then delivered to a post-

[11] www.speak4it.com

Figure 9: iPizza - multimodal pizza ordering prototype - initial screen and shopping cart example

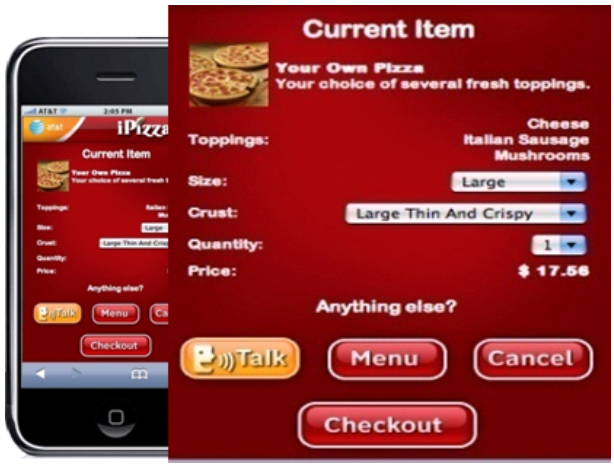

Figure 10: iPizza - multimodal pizza ordering prototype - ordered item editing

processing URL where an application web service sends in turns the speech recognition results to the local business search engine and to the map location service. Then it combines the resulting business listing and the mapping information into a format suitable for the visualization capabilities of the device and returns the response to the client though the same HTTP request.

8. NETWORK LATENCY EVALUATION

In order to evaluate the impact of the network latency on the user experience, we instrumented the Java ME client available on the speech mashup portal to measure the following experimental parameters:

- the **RF signal power level** measured just before the data transmission started. This gives an indication of the strength of the radio signal from the cell tower. If the signal is weak, the exchange of data packets slows down due to the highest packet retransmission rate (e.g., longer packet round-trip time). We assumed that the power level remained substantially stable for the duration of the input utterance.
- the **network connection time (CT)**. This is the time required by the mobile phone to open an HTTP connection and it is measured from the instant the open command is issued to the time the socket connection is established and ready to transmit data.
- the **overall response time (RT)**. This is the time elapsed between the last audio packet of audio transmission and the time the response from the speech recognizer is received. It gives an overall measure of the server processing time and the uplink transmission time. The speech recognition processing time did not affect this measure since all utterances were processed in real time.

Table 3: Latency measures on a 3G (EDGE) data network (CT: network connection time in sec, RT: overall response time in sec, $\bar{t}$: sample average time, s: sample standard deviation)

Bars	Power Level (dBm)	Samples	CT (sec) $\bar{t}$ / s	RT (sec) $\bar{t}$ / s
5	> -76	54	0.006 / 0.003	0.426 / 0.633
4	> -87 < -75	20	0.006 / 0.003	0.400 / 0.598
3	> -93 < -86	11	0.007 / 0.002	0.727 / 0.786
2	> -102 < -92	8	0.006 / 0.002	1.750 / 0.886

We ran the probing software client on a BlackBerry Curve 8320 with 3G network connectivity and sampled around one hundred speech recognition interactions equally divided into three different environmental conditions: 1) on a highway moving at 50-65 mph; 2) in an office environment; 3) on the lower-level floor of a building and away from the cell tower. We used a large grammar containing the US city/state pairs with 35,987 entries and a vocabulary of 22,527 words. With an average utterance duration of 2.95 s, the amount of data streamed over the wireless network was around 4.39 kBytes.

The initial set of experiments showed a substantial delay in creating the data connectivity link. To open a TCP socket connection and start sending speech samples through HTTP, required between 0.5 and 3 s. We learned that the first step in establishing any data connectivity in a 3G network is the creation of a Packet Data Protocol (PDP) context. The PDP context assignment validates the mobile subscriber's session and creates an IP address for the device by engaging several exchanges with the cell tower. This is a time-consuming step, but once a PDP context has been created, it exists until data traffic ends or some timeout expires (i.e., there is some timeout between the end of the last TCP session teardown and possibly a different timeout after a TCP packet is sent or received).

To reduce the PDP context creation time, we allocated a pool of socket connections during the application startup time. Making the active pool of sockets immediately available to the application for data transmission reduced drastically the initial connectivity time. Latency measures in table 3 show an average connection time of 6 ms across different signal power conditions. The response time exhibits more variability across different RF signal conditions with around 0.5 s delay in most of the cases and a maximum value of 3.77 s when the signal power degrades significantly and packet retransmission rate increases. Overall, the resulting latency is within acceptable limits, but further studies are required to determine how the latency would affect the user satisfaction.

9. RELATED WORK

There are similar works published in the recent literature such as [1, 10, 5] addressing vertical applications for voice search that are based on *ad hoc* architectures. The WAMI (Web-Accessible Multimodal Applications) toolkit [17] proposes a general framework for multimodal applications and it is closer to our model. However, some features are either partially addressed or missing. For example, compared to WAMI, the speech mashups offers a complete range of configurations (Section 4) and full support for a variety of web and speech standards. Moreover, the speech mashup framework implements fully scriptable portal APIs, stochastic language models support, HTTP chunking, TTS bookmarking, and it

has been tested on an industrial-strength cloud computing system. Other interesting approaches described in [23, 8, 22] try to reuse the existing VoiceXML infrastructure and expand it with multimodal interaction based on a more traditional telephony model.

10. CONCLUSIONS

The speech mashup architecture proposes a general framework for *mashing up* speech and web services in a multimodal mobile environment. It provides a web service-based access to speech processing resources (ASR and TTS) in the *"cloud"* and different architecture configurations to minimize network latencies and increase security. The web-based portal grants easy access to grammar tools and service administration. Latency measures in a 3G network show very promising results, although more experiments should be conducted to evaluate other networks and devices. Access to the platform is available for noncommercial use by registering an account at `https://service.research.att.com/smm/`.

11. ACKNOWLEDGMENTS

We thank Alan Glasser and Jack Lucas for their invaluable architecture insights, Danilo Giulianelli for setting up the mashups on the Amazon EC2 cloud, Linda Crane for her precious feedback and patience, and the anonymous reviewers for their comments on the early version of this paper.

References

[1] A. Acero, N. Bernstein, R. Chambers, Y. C. Jui, X. Li, J. Odell, P. Nguyen, O. Scholz, and G. Zweig. Live search for mobile: Web services by voice on the cellphone. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

[2] C. Allauzen, M. Mohri, and B. Roark. The design principles and algorithms of a weighted grammar library. *International Journal of Foundations of Computer Science*, 16:403–421, 2005.

[3] G. Alonso, F. Casati, H. Kunoand, and V. Machiraju. *Web Services*. Springer, November 2003.

[4] J. Axelsson, C. Cross, H. W. Lie, G. McCobb, T. V. Raman, and L. Wilson. XHTML+Voice Profile 1.0. World Wide Web Consortium, Note NOTE-xhtml+voice-20011221, December 2001.

[5] M. Bacchiani, F. Beaufays, J. Schalkwyk, M. Schuster, and B. Strope. Deploying goog-411: early lessons in data, measurement, and testing. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

[6] J. L. Beckham, G. Di Fabbrizio, and N. Klarlund. Towards SMIL as a foundation for multimodal, multimedia applications. In *Eurospeech 2001, European Conference on Speech Communication and Technology*, Aalborg, Denmark, September 3-7 2001.

[7] T. Berners-Lee, R. T. Fielding, and H. Frystyk Nielsen. Hypertext transfer protocol — http/1.0. Internet RFC 1945, May 1996.

[8] E. Blechschmitt and C. Strödecke. An architecture to provide adaptive, synchronized and multimodal human computer interaction. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 287–290, New York, NY, USA, 2002. ACM.

[9] D. C. Burnett, M. R. Walker, and A. Hunt. Speech synthesis markup language (SSML) version 1.0. W3C recommendation, W3C, Sept. 2004. http://www.w3.org/TR/2004/REC-speech-synthesis-20040907/.

[10] S. Chang, S. Boyce, K. Hayati, I. Alphonso, and B. Buntschuh. Modalities and demographics in voice search: learnings from three case studies. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

[11] D. Crockford. The application/json media type for javascript object notation (json). RFC 4627, July 2006.

[12] B. Eberman, J. Carter, D. Meyer, and D. Goddeau. Building VoiceXML browsers with OpenVXI. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 713–717, New York, NY, USA, 2002. ACM.

[13] J. Feng and S. Bangalore. Query parsing for voice-enabled mobile local search. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2009.

[14] J. Feng, S. Banglore, and M. Gilbert. Role of natural language understanding in voice local search. In *10th Annual Conference of the International Speech Communication Association (Interspeech 2009)*, 2009.

[15] R. T. Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[16] V. Goffin, C. Allauzen, E. Bocchieri, D. Hakkani-Tür, A. Ljolje, S. Parthasarathy, M. Rahim, G. Riccardi, and M. Saraclar. The AT&T WATSON speech recognizer. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2005.

[17] A. Gruenstein, I. McGraw, and I. Badr. The WAMI toolkit for developing, deploying, and evaluating web-accessible multimodal interfaces. In *ICMI '08: Proceedings of the 10th international conference on Multimodal interfaces*, pages 141–148, New York, NY, USA, 2008. ACM.

[18] M. Johnston. EMMA: Extensible MultiModal Annotation markup language. Recommendation, W3C, February 2009. http://www.w3.org/TR/emma/.

[19] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 330–337, New York, NY, USA, 2006. ACM.

[20] S. McGlashan and A. Hunt. Speech recognition grammar specification version 1.0. W3C recommendation, W3C, Mar. 2004. http://www.w3.org/TR/2004/REC-speech-grammar-20040316/.

[21] E. Newcomer and G. Lomow. *Understanding SOA with Web Services (Independent Technology Guides)*. Addison-Wesley Professional, 2004.

[22] G. Niklfeld, R. Finan, and M. Pucher. Architecture for adaptive multimodal dialog systems based on VoiceXML. In *Proceedings of EuroSpeech '01*, 2001.

[23] G. Niklfeld, M. Pucher, R. Finan, and E. Eckhart. Mobile multi-modal data services for GPRS phones and beyond. In *Fourth IEEE International Conference on Multimodal Interfaces*, pages 337– 342, 2002.

[24] B. Porter, S. McGlashan, D. Burke, D. C. Burnett, E. Candell, R. Auburn, P. Baggia, J. Carter, K. Rehor, M. Oshry, M. Bodell, and A. Lee. Voice extensible markup language (VoiceXML) 2.1, June 2007. http://www.w3.org/TR/2007/REC-voicexml21-20070619/.

[25] S. Shanmugham and D. Burnett. Media Resource Control Protocol Version 2 (MRCPv2), May 5 2009.

[26] K. Wang. SALT: an XML application for web-based multimodal dialog management. In *NLPXML '02: Proceedings of the 2nd workshop on NLP and XML*, pages 1–8, Morristown, NJ, USA, 2002. Association for Computational Linguistics.

[27] R. Yee. *Pro Web 2.0 Mashups: Remixing Data and Web Services (Profesional Reference Series)*. APress, 2008.