Catch This SpeechEvent— Recognition and Synthesis on Devices

Charles Hemphill, Senior Speech Scientist

Steve Rondel, CEO

Conversay
www.conversay.com

TS-5932

Goal

What you will learn

Learn how to effectively add speech recognition and speech synthesis to your applications to make them faster, easier, and more fun to use.

Agenda

Background and motivation

Design considerations

Programming examples

The TCK

Adoption of the API

Conclusions and directions

Background and Motivation

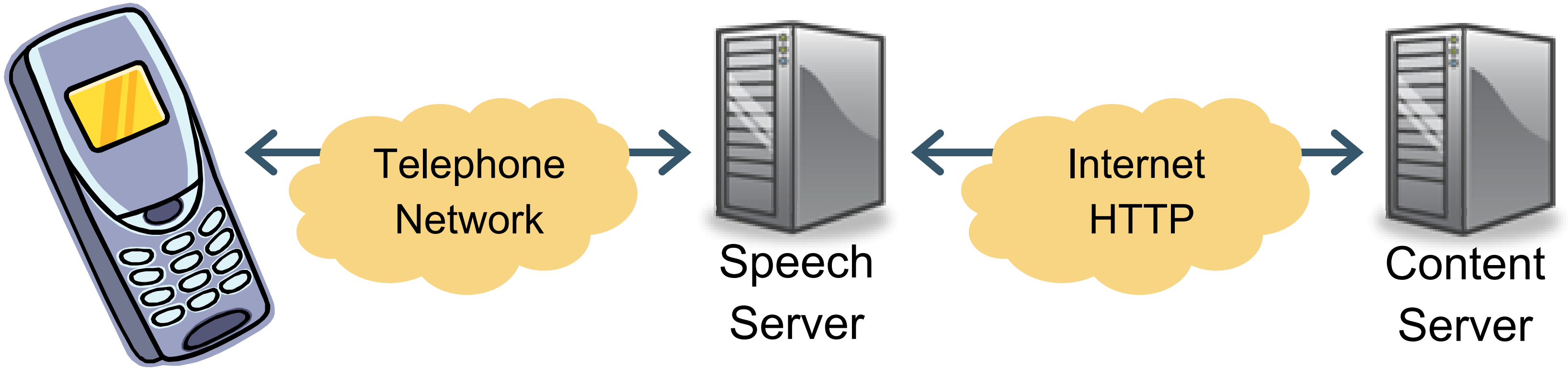
The explosion of the embedded world

- Billions of Java™ Platform, Micro Edition (Java ME platform) devices
- Network in your pocket
  - Lots of content
- Smaller form factors
  - Reduced screen sizes
  - Limited number or size of buttons
  - Can be harder to use

Background and Motivation

Server-side speech approaches

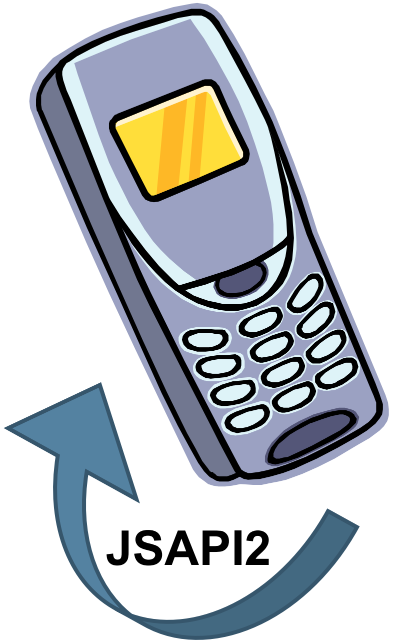
- Speech technology on a server
- Can be used with standard telephones
- Uses the W3C VoiceXML markup language



Background and Motivation

Client-side speech approaches

- Speech technology on the device
- Fits well with Java ME platform devices
  - Enough CPU and memory
- Increases usability of the device
- Minimizes latency for responses
- Easier access to local resources
  - Screen, user data, and local applications
- Supports multi-modal interaction
- No telephony channel noise



Background and Motivation

Many applications for speech **recognition** and **synthesis**

- Games—“Fire missiles at the red team in sector 7”
- Data entry—“39 widgets left in bin 27”
- E-mail—“Read the message from Steve”
- Calendar—“Is next Thursday open?”
- Learning—“What is 8 times 7?”
- Accessibility—“Read chapter 3”
- Car navigation—**“Take exit 34 in ½ mile”**
- System Alerts—**“Your fuel level is low”**

Background and Motivation

We need a standard

- Java Speech API 2 (JSAPI2)—Java Specification Request (JSR) 113
- Based on JSAPI1 for Java Platform, Standard Edition (Java SE)
- Aimed at Java ME platform
- Covers both recognition and synthesis
- Makes speech technologies easy to use
- Expert Group participants
    - Andrew Thompson, **Conversay (specification lead)**, IBM, Intel, Nokia, Motorola, Sun, Texas Instruments

Agenda

Background and motivation

Design considerations

Programming examples

The TCK

Adoption of the API

Conclusions and directions

Design Considerations

Fitting speech onto the Java ME platform

- Speech recognition
  - Name and number dialing
  - Built-in grammars (device specific)
  - Application-defined grammars
- Speech synthesis
  - Formant synthesis
  - Concatenative synthesis
- Does not include:
  - Explicit dictation support
  - Speaker verification
  - Speaker identification

Design Considerations

Basic building blocks for Java SE platform compatibility

- Add almost nothing for Java SE platform compatibility
- Language selection is important
  - Locale–8 methods for language, country, and variant
- The API is event driven
  - EventListener—a simple tagging interface
  - EventObject—constructor, getSource, toString
- Speech engines have many knobs
  - PropertyChangeListener—one method interface
  - PropertyChangeEvent—three methods

Design Considerations

Adopting standards from the W3C

- Speech Synthesis Markup Language (SSML)

```
<speak> Java <emphasis>talks</emphasis> now!
</speak>
```

- Speech Recognition Grammar Specification (SRGS)—XML format only

```
<grammar> <rule id="yes-or-no">
    <one-of>
        <item>yes</item> <item>no</item>
    </one-of>
</rule> </grammar>
```

Design Considerations

Integrating speech and GUI events

- JSAPI1 relied on AWT—no Applets ☹
- SpeechEventExecutor interface for JSAPI2
  - Compatible with JSR 116 (Executor mechanism)
  - Can integrate events with lcdui, Swing, etc.

```
// Put SpeechEvents on the MIDlet’s UI thread
engine.setSpeechEventExecutor(
  new SpeechEventExecutor() {
    public void execute(Runnable r) {
    javax.microedition.lcdui.Display.getDisplay(
      this).callSerially(r);
}});
```

Design Considerations

Defining audio input and output with an AudioManager

- AudioManager interface supports media locators
- Can be implemented with JSR 135
- Can define input and output sources

```
Synthesizer synth = ... // more detail later
// Can throw SecurityException or AudioException
AudioManager am = synth.getAudioManager();
am.setMediaLocator("file:///user/smith/hello.wav");
```

- Supports addAudioListener for AUDIO_LEVEL and other AudioEvents

Design Considerations

Security

- Security mechanisms provided by the underlying profile and configuration (e.g., MIDP2)
- An implementation must guarantee that:
  - SecurityException is thrown when the caller does not have the appropriate security permissions
  - The method can be used when the appropriate permissions are granted
- System properties determine permission
  - Method: javax.speech.AudioSegment.getInputStream
  - Key: javax.speech.supports.audio.capture

Agenda

Background and motivation

Design considerations

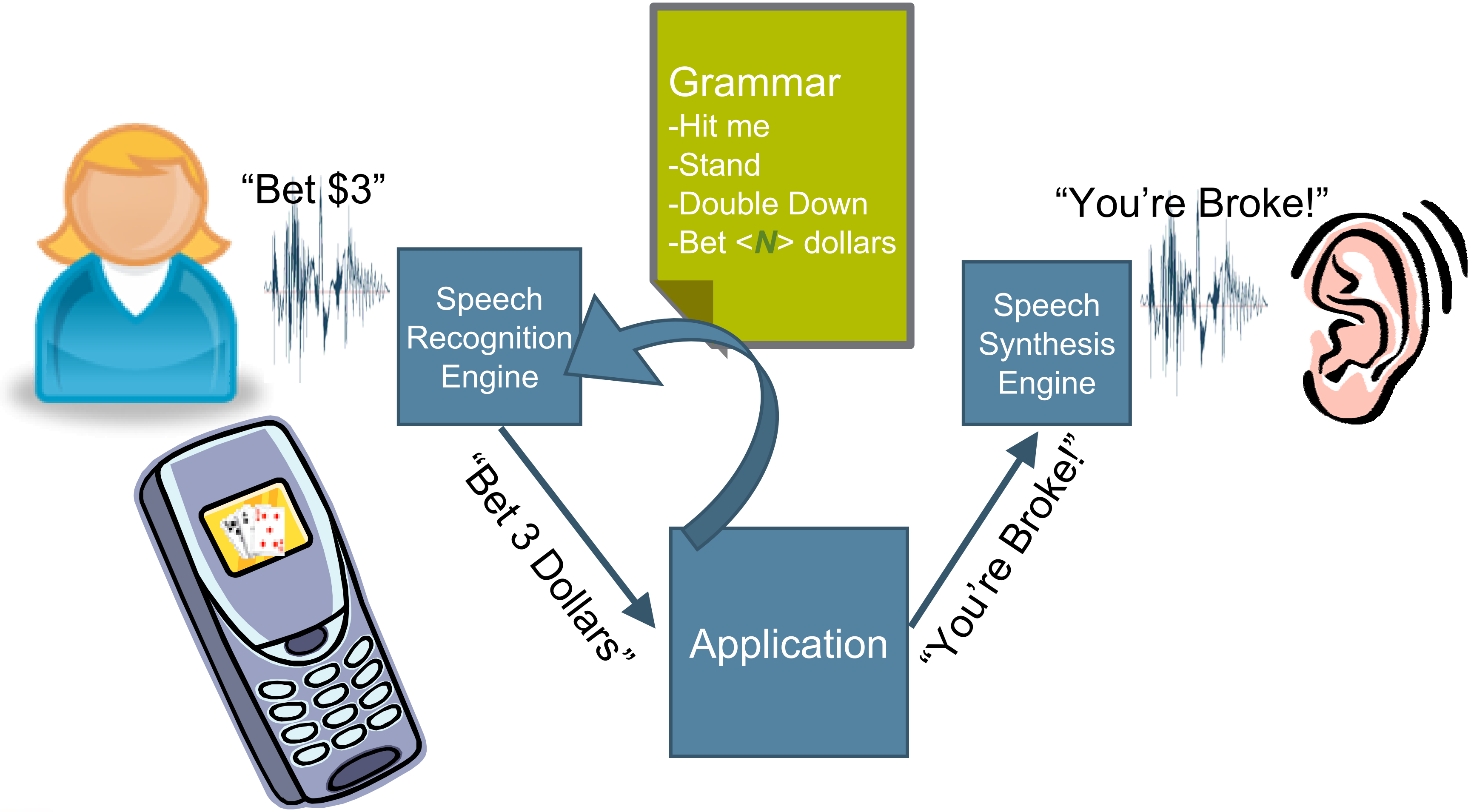
Programming examples

The TCK

Adoption of the API

Conclusions and directions

A Simple Conversation Example



“Hello World” for Synthesis

```
// Create a synthesizer for the default Locale
Synthesizer synth = (Synthesizer)
   EngineManager.createEngine(SynthesizerMode.DEFAULT);

// Load language specific data - can take time
synth.allocate();

// Speak the "Hello world" string
synth.speak("Hello, world!", null);

// Clean up - includes waiting for the queue to empty
synth.deallocate();
```

“Hello World” for Recognition (1/3)

```
import javax.speech.*;
import javax.speech.recognition.*;

public class HelloWorld implements ResultListener {
  static Recognizer rec;
  static final String grammarMarkup =
    "<grammar root='s' xml:lang='en' version='1.0'
              xmlns='http://www.w3.org/2001/06/grammar'>" +
      "<rule id='s' scope='public'>" +
        "<one-of>" +
          "<item> hello world </item>" +
          "<item> hello computer </item>" +
        "</one-of>" +
      "</rule>" +
    "</grammar>";
  ...
```

“Hello World” for Recognition (2/3)

```
public static void main(String args[]) {// try/catch omitted
  // Create a recognizer for the default Locale
  Recognizer rec = (Recognizer)
      EngineManager.createEngine(RecognizerMode.DEFAULT);

  // Load language specific data – can take time
  rec.allocate();

  RuleGrammar gram =           // what to recognize
      rec.loadRuleGrammar("HelloWorld.s", grammarMarkup);

  rec.addResultListener(this); // get recognized words
  rec.requestFocus();          // user talks to us
  rec.resume();                // process audio

  // Would do other things here - wait for deallocate
  rec.waitEngineState(Engine.DEALLOCATED);
}
```

“Hello World” for Recognition (3/3)

```
// Receive RESULT_ACCEPTED event: print it, clean up
public void resultUpdate(ResultEvent event) {
  if (event.getId() == RESULT_ACCEPTED) {
    try {
      Result r = (Result) (event.getSource());
      ResultToken tokens[] = r.getBestTokens();
      for (int i = 0; i < tokens.length; i++)
        System.out.print(tokens[i].getSpokenText() + " ");
      System.out.println();
      // For this example, deallocate the recognizer
      rec.deallocate();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

DEMO

Hello World

Speech-Enabled Blackjack

MIDlet Examples

Grammars From Buttons

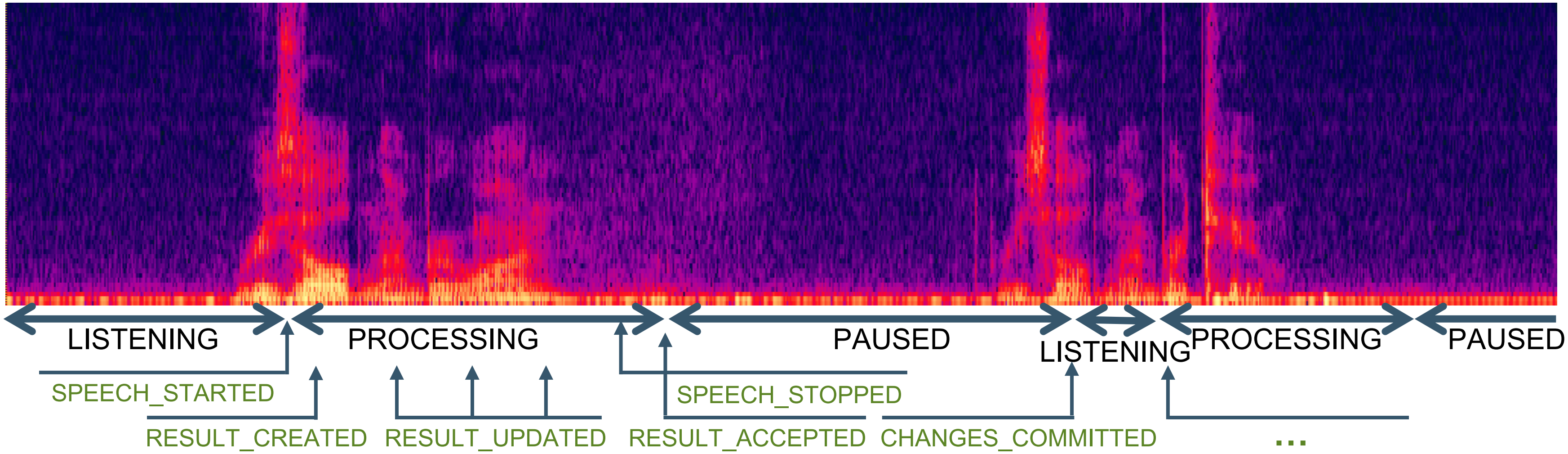
```
Vector buttons = ... // from the application
String grammarName = “blackjack_grammar_” + (grammarID++);
RuleGrammar grammar =
    recognizer.createRuleGrammar(grammarName, “start”);

RuleComponent[] alts = new RuleComponent[buttons.size()];

for (int i = 0; i < alts.length; i++) {
    RuleToken token =
        new RuleToken(buttons.elementAt(i).toString());
    RuleTag tag = new RuleTag(String.valueOf(i));
    alts[i] = new RuleSequence(
        new RuleComponent[] { token, tag });
}

grammar.addRule(
    new Rule(“start”, new RuleAlternatives(alts),
             Rule.PUBLIC_SCOPE));
```

Catching SpeechEvents



- API “does the right thing” in basic cases
- Applications can use these events to:
  - Cancel synthesis
  - Update the display
  - Change grammars

Agenda

Background and motivation

Design considerations

Programming examples

The TCK

Adoption of the API

Conclusions and directions

The TCK

Architecture

- Prevents fragmentation by ensuring that the RI implements the specification
- Based on Sun's Test Development Kit (TDK)
- MIDP2 emulator used as the test agent
- Uses a standard web server (Tomcat)
  - Reports test results for assertions
  - Supports testing directly on devices
- Runs semi-automatically on the desktop
  - Some user feedback required for synthesis decisions

The TCK

Assertions

- Assertions from the W3C
  - Over 200 for SSML
  - Over 150 for SRGS
- Hundreds of additional JSAPI2-specific assertions
- SSML example with manual confirmation

```
<speak version="1.0" xml:lang="en-US"
           xmlns=http://www.w3.org/2001/10/synthesis>
    This specification is from the
    <say-as interpret-as="letters"> w3c </say-as>.
</speak>
```

DEMO

TCK Assertions

Agenda

Background and motivation

Design considerations

Programming examples

The TCK

Adoption of the API

Conclusions and directions

Adoption of the API

Developers

- Aimed at devices
- JSAPI2-enabled emulator runs on desktops
- Making a development kit available
  - Specification, Reference Implementation (RI)
  - Examples
- Encouraging compelling applications
- Building an application suite for reference

Adoption of the API

Tools and other standards

- Talking to the Sun Java Wireless toolkit team for inclusion
  - Want to reach developers
  - Easier to integrate with other JSRs
  - “Should have been there from the beginning”
- Working to include JSAPI2 in the next umbrella JSR, JSR 249 (Mobile Service Architecture Advanced)
  - Speech is a natural user interface
  - Speech is an aid for those with disabilities

Agenda

Background and motivation

Design considerations

Programming examples

The TCK

Adoption of the API

Conclusions and directions

Conclusions and Directions

Next steps

- WTK, JSR 249, and other adoption efforts
- Incorporate feedback from developers
- Support a developer network
- Consider API improvements
  - More support for large-vocabulary recognition
  - SpeakerProfile management
- Upgrade the RI technology
  - Larger vocabulary
  - More natural-sounding synthesis

Conclusions and Directions

Acknowledgements

- Many thanks to the Expert Group participants
  - Andrew Thompson, IBM, Intel, Nokia, Motorola, Sun, Texas Instruments
- Technical staff at Sun
- Java Community Process^{SM} program office
- Conversay team members

Summary

- JSR 113 is an API for speech-enabling mobile applications
- Encompasses a range of speech technologies
- Improves the user interface
- Is easy to use and incorporate into applications
- Supports applications that might otherwise be impractical

For More Information

See:

- Exhibit booth
- www.conversay.com
- W3C standards
  - SSML: http://www.w3.org/TR/speech-synthesis/
  - SRGS: http://www.w3.org/TR/speech-grammar/
- JSR references
  - 113, 116, 135, 249
  - www.jcp.org

Q&A

Charles Hemphill and Steve Rondel

Catch This SpeechEvent—Recognition and Synthesis on Devices

Charles Hemphill, Senior Speech Scientist

Steve Rondel, CEO

Conversay
www.conversay.com

TS-5932