COMP349
Spoken Language Dialog Systems
Speech Synthesis ML & Semantic Interpretation

Rolf Schwitter
schwitt@ics.mq.edu.au
Today’s Program

• Speech Synthesis Markup Language (SSML)
• Speech Markup Elements
• Speech Synthesis Process
• Semantic Interpretation for Speech Recognition (SISR)
• Processing Semantic Interpretation Tags
• Mapping Semantic Results
Recap: The Bigger Picture
Text To Speech

Modifying the Synthesis

- Sometimes we want to change the way speech sounds.
- Comas help to get better phrasing, especially in long sentences.
- Speech can be controlled via markup, for example:
  
  - `<prosody rate="slow"> this is speaking slowly </prosody>`.
  - Break for 3 seconds `<Break time="3s"/>` Okay, keep going.
  - `<voice name="crystal">Crystal, 1 2 3.</voice>`
    `<voice name="mike">Mike.</voice>`
    `<voice name="rosa">Rosa, 1 2 3.</voice>`
    Back to Mike.
  
  `</voice>`
Speech Synthesis Markup Language (SSML)

- SSML aims at
  - improving the quality of synthesized content and
  - providing control across platforms and SS implementations.
- SSML markup may be
  - present within a complete SSML document or
  - embedded in another language (e.g. VoiceXML).
- SSML markup can be used to model the content of the `<prompt>` element in VoiceXML.
A Complete SSML Document

• Example of an SSML document for an email reader:

```xml
<?xml version="1.0"?>
<!DOCTYPE speak PUBLIC "-//W3C//DTD SYNTHESIS 1.0//EN"
  "http://www.w3.org/TR/speech-synthesis/synthesis.dtd">  
<speak version="1.0"
  xmlns="http://www.w3.org/2001/10/synthesis"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
  http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
  xml:lang="en-US">
  <p>
    <s>You have 4 new messages.</s>
    <s>The first is from Stephanie Williams and arrived at <break/>3:45pm.</s>
    <s>The subject is <prosody rate="-20%">ski trip</prosody></s>
  </p>
</speak>
```
Explanation

• An SSML document is an XML application.
• The root element is <speak>.
• <p> and <s> are used to mark the text structure.
• <break> has the effect of marking the time as important.
• <prosody> is used to slow the speaking rate (-20%).
<prompt>
  <emphasis>Welcome</emphasis>
  to the Bird Seed Emporium.
  <audio src="rtsp://www.birdsounds.example.com/thrush.wav"/>
  We have
  <say-as interpret-as="number">250</say-as>
  kilogram drums of thistle seed for
  <say-as interpret-as="currency">$299.95</say-as>
  plus shipping and handling this month.
  <audio src="http://www.birdsounds.example.com/mourningdove.wav"/>
</prompt>
Explanation

- The `<say-as>` element helps for rendering the contained text.
- The "interpret_as" attribute indicates the content type.
- A synthesis processor should be able to support the most common formats (dates, times, acronyms, etc.).
- When the value for the "interpret_as" is unknown, the synthesis processor should not fall over.
# Speech Markup Elements in VoiceXML 2.0

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;audio&gt;</code></td>
<td>Specifies audio files to be played and text to be spoken.</td>
</tr>
<tr>
<td><code>&lt;break&gt;</code></td>
<td>Specifies a pause in the speech output.</td>
</tr>
<tr>
<td><code>&lt;desc&gt;</code></td>
<td>Provides a description of a non-speech audio source in <code>&lt;audio&gt;</code>.</td>
</tr>
<tr>
<td><code>&lt;emphasis&gt;</code></td>
<td>Specifies that the enclosed text should be spoken with emphasis.</td>
</tr>
<tr>
<td><code>&lt;lexicon&gt;</code></td>
<td>Specifies a pronunciation lexicon for the prompt.</td>
</tr>
<tr>
<td><code>&lt;mark&gt;</code></td>
<td><em>Ignored by VoiceXML platforms.</em></td>
</tr>
<tr>
<td><code>&lt;meta&gt;</code></td>
<td>Specifies meta and &quot;http-equiv&quot; properties for the prompt.</td>
</tr>
<tr>
<td><code>&lt;metadata&gt;</code></td>
<td>Specifies XML metadata content for the prompt.</td>
</tr>
<tr>
<td><code>&lt;p&gt;</code></td>
<td>Identifies the enclosed text as a paragraph, containing zero or more sentences.</td>
</tr>
<tr>
<td><code>&lt;phoneme&gt;</code></td>
<td>Specifies a phonetic pronunciation for the contained text.</td>
</tr>
<tr>
<td><code>&lt;prosody&gt;</code></td>
<td>Specifies prosodic information for the enclosed text.</td>
</tr>
<tr>
<td><code>&lt;say-as&gt;</code></td>
<td>Specifies the type of text construct contained within the element.</td>
</tr>
<tr>
<td><code>&lt;s&gt;</code></td>
<td>Identifies the enclosed text as a sentence.</td>
</tr>
<tr>
<td><code>&lt;sub&gt;</code></td>
<td>Specifies replacement spoken text for the contained text.</td>
</tr>
<tr>
<td><code>&lt;voice&gt;</code></td>
<td>Specifies voice characteristics for the spoken text.</td>
</tr>
</tbody>
</table>
Example: `<sub>` Element

- `<sub>` indicates that the value of the "alias" attribute replaces the contained text for pronunciation:

```xml
<?xml version="1.0"?>
<Speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                           http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
       xml:lang="en-US">
  <sub alias="World Wide Web Consortium">W3C</sub>
  <!-- World Wide Web Consortium -->
</Speak>
```

- This allows a document to contain both a written and spoken form.
Speech Synthesis Process

- Processing steps to convert marked-up text into voice output are:

  - XML Parse
  - Structure Analysis
  - Text Normalization
  - Text-to-Phoneme Conversion
  - Prosody Analysis
  - Waveform Production
XML Parse

• An XML parser extracts the document tree and content from the incoming marked-up text document.
• The structure, attributes and tags influence the subsequent steps.
• Note: Tokens in SSML cannot span mark-up tags.
• For example, the synthesizer will treat
  "cup <break/> board"
as the two words
  "cup" and "board"
rather than as one word with a pause in the middle.
Structure Analysis

• The document structure influences how a document should be read.
• There are common speaking patterns associated with paragraphs and sentences.
• The <p> and <s> elements indicate document structures that affect the speech output.
• Without these elements, the synthesizer would have to infer the structure relying on punctuation and other language-specific data.
Text Normalization

• All written languages have constructs that require a conversion of the written form into the spoken form.

• For example, in English:
  – "$200" may be spoken as "two hundred dollars";
  – "1/2" may be spoken as "half";
  – "January 2nd" may be spoken as "January second";
  – "10-4" may be spoken as "ten four", "ten minus four", "forth of October", "tenth of April".
Text Normalization

- The <say-as> element can be used to indicate the type of such constructs and to resolve ambiguities.
- The set of constructs that can be marked has not yet been defined.
- Possibilities are: dates, times, numbers, acronyms, and currency.
- However, many acronyms and abbreviations can be handled via direct text replacement.
- You can use the <sub> element:
  - "BBC" can be written as "B B C"
  - "AAA" can be written as "triple A".
If you have a synthesizer that supports both Kanji and Kana, you may be able to use the `<sub>` element to identify whether 今日は should be spoken as きょうは ("kyou wa" = "today") or こんにちは ("konnichiwa" = "hello").
Text-to-Phoneme Conversion

- Once the synthesizer has determined the set of words to be spoken, it must derive pronunciations for each word.
- Word pronunciations can be described as sequences of phonemes.
- **Phonemes** are units of sound in a language that distinguish one word from another (e.g. /cat/ versus /rat/).
- Most English dialects have around 45 phonemes.
- Problem: differences between written and spoken forms.
Text-to-Phoneme Conversion

• In English:
  – "read" may be spoken as "reed" (I will read the book) or
  – "red" (I have read the book).

• Problems with non-English-origin names:
  – "Caius College" (pronounced "keys college"),
  – The president of the Republic of Kiribati (pronounced "kiribass").
Text-to-Phoneme Conversion

• Support in SSML:
  – `<phoneme>` element allows a phonemic sequence to be provided for any word or word sequence.
  – `<say-as>` element might be used to indicate the proper form of the text so that the synthesizer can apply special rules to determine a pronunciation.
  – `<lexicon>` element can be used to reference external definitions of pronunciations.
Text-to-Phoneme Conversion

- Synthesizers should support the “IPA“ alphabet.
- That’s an Unicode\(^1\) representations of the phonetic characters developed by the International Phonetic Association.
- Example:

```xml
<phoneme
  alphabet = "ipa"
  ph = "t\&#x259;mei\&#x325;\&#x27E;ou\&#x325;">
  tomato
</phoneme>
```

\(^1\) 16-bit character set that assigns unique character codes to characters in a wide range of languages. Unlike ASCII, which defines 128 distinct characters typically represented in 8 bits, there are as many as 65,536 distinct Unicode characters that represent the unique characters used in many languages.
Prosody Analysis

• Prosody permits control for a set of features for speech output:
  – pitch (also called intonation or melody),
  – timing (or rhythm),
  – pausing,
  – speaking rate,
  – emphasis on words, etc.

• Producing human-like prosody is important
  – for making speech sound natural and
  – for correctly conveying the meaning of spoken language.
Prosody Analysis

• A synthesizer can be supported in generating appropriate prosodic features with the help of the `<emphasis>`, `<break>` and `<prosody>` elements.

• Example:

```xml
<prompt>
    The price of XYZ is <prosody rate="-10%">$45</prosody>
</prompt>
```

• Since voices are processor-specific, the default rate will vary.
Wave Form Production

- Phonemes and prosodic information are used by the synthesizer to produce the audio waveform.
- The `<voice>` element allows for requesting
  - a particular voice or
  - specific voice qualities (e.g. a young male voice).
- Additionally, the `<audio>` element allows for insertion of recorded audio data into the output stream.
Wave Form Production

• Example (<voice> element):

```xml
<voice gender="female">Mary had a little lamb,</voice>
<!-- now request a different female child's voice -->
<voice gender="female" variant="2">Its fleece was white as snow.</voice>
<!-- processor-specific voice selection -->
<voice name="Mike">I want to be like Mike.</voice>
```
Again: The Bigger Picture
Semantic Interpretation for Speech Recognition

- The SISR language defines the syntax and semantics of the contents of the "tag" attribute in SRGS.
- Knowing the sequence of words that were uttered is interesting but often not the most practical way of handling the information.
- What is needed is a processable representation of the information.
- SI tags provide a means to attach instructions for the computation of semantic results to a SR grammar.
A Simple Example

<grammar mode = "voice" root = "destination-city">
  <rule id = "destination-city">
    <one-of>
      <item> New York </item>
      <item tag = "New York"> Big Apple </item>
      <item> Washington </item>
      <item tag = "Washington"> The Capital </item>
    </one-of>
  </rule>
</grammar>
Processing Semantic Interpretation Tags

• It is possible that a grammar will match but not return a semantic interpretation.

• In this case, the platform will use the raw text string for the utterance as the semantic result.

• When used with a VoiceXML processor,

  it is expected that a SI tag processor
  will convert the result generated by an SRGS processor
  into an ECMAScript object.
Rule Variables

• Every grammar rule has a single rule variable that holds a value.
• The rule variable is identified by “out”.
• Properties of the rule variable can accessed by “out.identifier”
  where “identifier” is the name of the property.
• For example:

  out (identifies the Rule Variable)
  out.pizza (identifies the pizza property of the Rule Variable)
Rule Variables

- The rule variable is initialized to an **empty object** before the first tag in the grammar rule is executed.
- **No** var statement is required.
- Properties can be added to this object.
- Individual properties of a rule variable can be identified by
  ```
  rules.rulename.identifier
  ```
  where “rulename” is the name of the rule and “identifier” is the name of the property.
Examples

// The Rule Variable associated to the referenced rule "ruleName"
rules.ruleName

// The property "prop" of the Rule Variable associated with the referenced rule "ruleName"
rules.ruleName.prop

// The Rule Variable associated to the latest matching rule reference before the SI Tag
rules.latest()

// The property "prop" of Rule Variable associated to latest matching rule reference before the SI Tag
rules.latest().prop
Example: Rule Variables

```xml
<grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar"
xml:lang="en-US" tag-format="semantics/1.0" root="drink">

<rule id="drink">
  <tag>rules.foodsize=medium;</tag>
  <item repeat="0-1">
    <ruleref uri="#foodsize"/>
  </item>
  <ruleref uri="#kindofdrink"/>
  <tag>out.drinksize=rules.foodsize;
         out.type=rules.kindofdrink;
  </tag>
</rule>
```
Example: Rule Variables

```xml
<rule id="foodsize">
  <one-of>
    <item>small</item>
    <item>medium</item>
    <item>large</item>
  </one-of>
</rule>

<rule id="kindofdrink">
  <one-of>
    <item>coke</item>
    <item>pepsi</item>
  </one-of>
</rule>
```

```xml
</grammar>
```
Example: Rule Variables

```
<rule id = "toppings" scope = "public">
  <tag> $ = new Array; </tag>
  <item repeat = "1-">
    <ruleref uri = "#topping"/>
    <tag> $.push($topping); </tag>
  </item>
</rule>

<rule id = "topping" scope = "public">
  <one-of>
    <item> cheese </item>
    <item> ham </item>
    <item> pepperoni </item>
    <item> eggs </item>
    <item> anchovies </item>
    <item> mushrooms </item>
  </one-of>
</rule>
```
Mapping Results

- Every `<field>` element has an associated slot name.
- The slot name is the value of the "slot" attribute.
- For example: `<field name = "from_city" slot = "from">`
- This name can be used to populate the variable.
- If the slot name is absent, it defaults to the variable name of the field!
- The "slot" attribute is useful if the grammar format has a mechanism for returning sets of attribute-value pairs and the slot names differ from the field variable names.
Example: VoiceXML (excerpt)

...  
  <field name = "from_city" slot = "from">
      <prompt> From which city are you leaving? </prompt>
      <grammar src = "cities.grxml#from"/>
  </field>

  <field name = "to_city" slot = "to">
      <prompt> To which city do you want to fly? </prompt>
      <grammar src = "cities.grxml#to"/>
  </field>
<rule id = "from" scope = "public">
    <item repeat = "0-1"> <ruleref uri = "#filler"/> </item>
    <item repeat = "0-1"> from </item>
    <ruleref uri = "#city"/> <tag> $.from = $city; </tag>
</rule>

<rule id = "to" scope = "public">
    <item repeat = "0-1"> <ruleref uri = "#filler"/> </item>
    <item repeat="0-1"> to </item>
    <ruleref uri = "#city"/> <tag> $.to = $city; </tag>
</rule>
<rule id = "filler">
  I want
  <item repeat = "0-1">
    to fly
  </item>
</rule>

<rule id = "city">
  <one-of>
    <item> London </item>
    <item> Prague </item>
    <item> Rome </item>
  </one-of>
</rule>
Mapping Form Level Results

- Grammars specified at the form-level produce a form-level result which may fill multiple input items simultaneously.

- For example the sentence
  
  *I would like a coca cola and three large pizzas with pepperoni and mushrooms.*

  might result in the following interpretation:

  ```javascript
  { drink: "coke", pizza: { number: "3", size: "large", topping: [ "pepperoni", "mushrooms" ] } }
  ```
Assigning Results

• The result from a form-level grammar can be assigned to various input items within a form, for example:

  <field name = "..." slot = "drink"/>  => "coke"
  <field name = "..." slot = "pizza"/>  => {number: "3",
                                           size: "large",
                                           topping: ["pepperoni",
                                                     "mushroom"]}

  <field name = "..." slot = "pizza.number"/>  => "3"
  <field name = "..." slot = "pizza.size"/>    => "large"
Take-Home Messages

• SSML aims at improving the quality of synthesized content.
• SSML can be used in SSML documents or in VXML documents.
• Most steps in the speech synthesis process can profit from SSML.
• The SIRG language defines the contents of the "tag" attribute in SRGS.
• The SI tag processor converts the result generated by an SRGS processor into an ECMAScript object.
• The result can be a string, an attribute-value pair, or a nested "knowledge object".