COMP349
Spoken Language Dialogue Systems
SALT

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

• Overview of SALT
• How SALT is Used
• Implementation Architecture
• Benefits of SALT
• Using only SALT Markup
• Beyond Pure SALT
• VoiceXML versus SALT
• The Future
SALT

- SALT (= Speech Application Language Tags)
  - is an extension of HTML
  - consists of a small set of XML elements (tags)
  - adds a powerful speech interface to Web pages.
- SALT can be used for both
  - voice-only browsers
  - multimodal browsers.
Who Developed SALT?

- The SALT spec (version 1.0) was developed by the SALT Forum
  - [http://www.saltforum.org/](http://www.saltforum.org/)
  and later contributed to the W3C
  - [http://www.w3.org/](http://www.w3.org/)
- The SALT Forum was founded by
  - Microsoft, Cisco, SpeechWorks, Philips, Comverse and Intel.
- Today: 70 companies and growing.
Hello "SALT"

    <head>
        <title> SALT Demo </title>
    </head>
    <body onload = "hello.Start()">
        <salt:prompt id = "hello">
            Hello World
        </salt:prompt>
    </body>
</html>
Explanation

• SALT tags have been added to the HTML document:
  - `<xmlns:salt>` defines a namespace,
  - `<salt:prompt>` defines a speech prompt.
• Document needs to be loaded in a SALT 1.0 compatible browser.
• Methods such as `start()` initiate SALT tags.
• It would say "Hello World" using a text-to-speech engine.
Overview of SALT

- The main top-level elements of SALT are
  - `<prompt ...>` for speech synthesis configuration and prompt playing
  - `<listen ...>` for speech recognizer configuration, recognition execution and post-processing, and recording
  - `<dtmf ...>` for configuration and control of DTMF collection
  - `<smex ...>` for general-purpose communication with platform components
• Simple TTS prompt

  <salt:prompt id = "Welcome">
    Welcome to your SALT application.
    What would you like to do?
  </salt:prompt>

• Pre-recorded audio

  <salt:prompt id = "RecordedPrompt">
    <content href = "welcome.wav"/>
  </salt:prompt>
**Prompt using external SSML**

```xml
<salt:prompt id = "ExternalContent">
  <content href = "Welcome.ssml"
    type = "application/ssml+xml"/>
</salt:prompt>
```

**Dynamic prompt**

```xml
<salt:prompt id = "DynamicPrompt">
  Did you say <value targetelement = "txtOption"
    targetattribute = "value"/>?
</salt:prompt>
```
Using `<listen>` for speech recognition:

```
<salt:listen id = "listenEmployeeName">
    <grammar src = "MyGrammar.grxml"/>
    <bind targetelement = "txtName"
        value = "//employee_name"/>
</salt:listen>
```

Note: once recognised "//employee_name" is bound to "txtName".
Using `<listen>` for speech recognition with function call:

```xml
<salt:listen id = "listenEmployeeName"
    onreco = "processEmployeeName">  
    <grammar src = "MyGrammar.grxml"/>  
</salt:listen>

<script>
  <![CDATA[
    function processEmployeeName() {
    ... }
  ]]>  
</script>
```
• Using <listen> for voice recording:

```xml
<salt:listen id = "recordMessage"
    onreco = "processMessage">
    <record beep = "true"/>
</salt:listen>

<script>
    <![CDATA[
        function processMessage() {
            ...
        }
    ]]>
</script>
```
Child Components of SALT

- In addition, there are several child components.
- The input element `<listen>` and `<dtmf>` contain
  - grammar controls `<grammar>`
  - binding controls `<bind>`.
- The `<listen>` element contains the facility
  - to record audio input `<record>`.
- All top-elements contain the platform config element `<param>`.
- Microsoft provides an `<audiometer>` element.
How SALT Is Used

• There are two major scenarios for the use of SALT:
  – multimodal applications
  – voice-only and telephony applications.
Multimodal

- Multimodal
  - adding SALT to a visual page (HTML, cHTML, WML)
  - speech-enabling controls
    - for “push-to-talk” form-filling scenarios
    - for more complex mixed initiative capabilities.
Multimodal

- Recognition may be started by a browser event (clicking on button).
- Activates a grammar of an input field.
- Binds the recognition result into that field.

```html
<!-- HTML -->
  ...
  <input name="txtBoxCity" type="text" />
  <input name="buttonCityListen" type="button" onClick="listenCity.Start();" />
  ...

  <!-- SALT -->
  <salt:listen id="listenCity">
    <salt:grammar name="g_city" src="/city.grxml" />
    <salt:bind targetelement="txtBoxCity" value="/city" />
  </salt:listen>
</html>
```
Telephony

• For SALT applications without a visual display, the application drives interaction with the user by prompting for required information.
• The HTML scripting and event model performs this function.
• Using scripting and the event model, the full control is available to developers for the management of prompt playing grammar activation and processing of recognition results.
Telephony

• Implementations of SALT are expected to provide scriptlets.
• Scriptlets handle common dialog processing task.
• For example, the RunAsk() function of the next example
  — activates prompts and recognition
  — until the values of the input fields are filled.
Telephony

```html
<body onload="RunAsk()">
<form id="travelForm">
   <input name="txtBoxOriginCity" type="text" />
   <input name="txtBoxDestCity" type="text" />
</form>

<!-- Speech Application Language Tags -->
<salt:prompt id="askOriginCity"> Where would you like to leave from? </salt:prompt>
<salt:prompt id="askDestCity"> Where would you like to go to? </salt:prompt>

<salt:listen id="recoOriginCity" onreco="procOriginCity()">
   <salt:grammar src="city.xml" />
</salt:listen>

<salt:listen id="recoDestCity" onreco="procDestCity()">
   <salt:grammar src="city.xml" />
</salt:listen>
```
Telephony

<!-- scripted dialog flow -->
<script>
    function RunAsk() {
        if (travelForm.txtBoxOriginCity.value==") {
            askOriginCity.Start();
            recoOriginCity.Start();
        } else if (travelForm.txtBoxDestCity.value==") {
            askDestCity.Start();
            recoDestCity.Start();
        }
    }

    function procOriginCity() {
        travelForm.txtBoxOriginCity.value = recoOriginCity.text;
        RunAsk();
    }

    function procDestCity() {
        travelForm.txtBoxDestCity.value = recoDestCity.text;
        travelForm.submit();
    }
</script>
</body>
</html>
Implementation Architecture

- Depending on the scenario, the following components may be necessary for implementing a SALT application:
  - a Web server
  - a telephony server
  - a speech server
  - a client device.
Web Server

• The Web server "generates" Web pages containing HTML, SALT, and embedded scripts.
• The script controls the dialog flow for voice-only interactions.
• For example, the script defines the order for playing audio prompts to a caller, assuming there are several prompts on a page.
Telephony Server

- Telephony server connects to the telephone network.
- The server incorporates:
  - a voice browser interpreter
    - interpreting the HTML, SALT, and script.
- The browser:
  - can run in a separate process for each caller
  - interprets only a subset of HTML
  - since much of the HTML refers to the GUI.
Speech Server

- The speech server
  - recognises speech
  - plays audio prompts
  - responses back to the user.
Client Device

- Clients include, for example,
  a pocket, a tablet, or desktop PC
  running a version of a browser (e.g. Internet Explorer)
  that is capable of interpreting HTML and SALT.
SALT Architecture
Benefits of SALT

• Benefits of using SALT:
  – reuse of application logic:
    the speech interface is a thin markup layer
    the code used for the business logic can be reused.
  – rapid development:
    mastering SALT is a rapid process
    the developer needs to learn very little extras.
  – speech + GUI:
    it is easy to create new multimodal applications.
Benefits of SALT

- Anybody wanting to speech-enable an application can use SALT.
- SALT markup is an interesting solution for adding speech to a Web page because it can leverage the scripting and event model inherent in HTML to implement the interactive flow with the user.
Example: Multimodal Application
Example: Multimodal Application

• This example shows the `<audiometer>` element in action:
Mixed Initiative

• Developers can add more complex mixed initiative capabilities.
• For example, a user can complete several forms with one utterance.
• A user can say:

  I want to fly to London on September 8.

• This places the correct information simultaneously in
  the city and date fields
  of the HTML page.
Example: Telephony Application

- This example uses Microsoft’s Speech Application SDK 1.0:
Example: Telephony Application

• Example dialog:

<table>
<thead>
<tr>
<th>Question</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>What type of coffee would you like?</td>
<td>(Mocha or Latte)</td>
</tr>
<tr>
<td>What size would you like?</td>
<td>(Tall or Grande)</td>
</tr>
<tr>
<td>Whole milk or non-fat?</td>
<td>(Whole milk or Non-fat)</td>
</tr>
</tbody>
</table>
Example: Telephony Application (Console)
Using Only SALT Markup

• In this scenario,
  use any text-based editor in any environment
to write application files with SALT markup,
including SALT attributes or properties,
calls to methods, and event handlers.
Using Only SALT Markup

• Download the Speech Application SDK 1.0: http://www.microsoft.com/speech/
• The Speech Application SDK contains a Speech Add-in.
• Client computers run the Web application using Microsoft Internet Explorer with the Microsoft Speech Add-in installed.
• The Speech Add-in comprises the DLLs that plug into Internet Explorer 6.0, which interpret SALT markup in HTML pages.
Using Only SALT Markup

• Important – if you write a SALT application:

  On each page of the application, insert a SALT namespace reference in the `<html>` element:


  The namespace declaration is necessary because all SALT elements have the "salt" prefix.
Deploying an Application

• Deploy the application on a Web server and test it.
• To deploy a speech-enabled Web application for access from IE:
  – set the MIME type on the Web server:
    \[ \text{mime-type} = \text{text/salt+html} \]
• Using Internet Information Services (IIS),
  – set the directory containing the application as an application root directory.
• Install the Speech Add-in for IE on each client computer.
Beyond Pure SALT

- The Microsoft Speech Application SDK (SASDK) is a set of development tools supporting the SALT specification.
- Authoring tools are integrated into Microsoft Visual Studio .NET 2003.
- SASDK provides
  - a set of ASP.NET speech controls
  - speech add-in for IE
  - debugging tools
  - tools for log analysis
  - grammar library.
VoiceXML versus SALT

• VoiceXML and SALT are both
  – markup languages
  – that describe speech interfaces.
• VoiceXML is designed for telephony applications:
  – interactive voice response applications are the focus.
• SALT targets speech application across a whole spectrum:
  – multimodal interactions are the focus.
VoiceXML versus SALT

The differences between VoiceXML and SALT are manifested in:
- the form of the markup
- the programming and execution model
- the level of programming interface available for developers.
Scope

- **VoiceXML**
  
  contains a large number of elements  
  since it defines a data and execution model  
  in addition to a speech interface.  

- **VoiceXML**
  
  deals not only with the user interface (e.g. `<prompt>`)  
  but also with data models (e.g. `<form>`, `<field>`) and  
  procedural programming (e.g. `<if>`, `<goto>`).
Scope

- SALT has only a handful of tags (e.g. `<prompt>`, `<listen>`) because it focuses on the speech interface.
- SALT does not define an execution model but instead uses existing execution models (HTML + JavaScript).
- SALT builds speech applications out of existing Web applications enables multimodal dialogs on a variety of devices.
Programming Model

• VoiceXML
  – contains a Form Interpretation Algorithm (FIA)
  – makes the finite-state nature of a dialog explicit.
• The FIA
  – applies field-filling control flow to the visiting fields within forms.
• The FIA
  can be manipulated and aborted via the use of
  conditional and procedural programming elements.
Programming Model

• SALT is based on an event-wiring model.
• Prompts and recognitions are activated on the basis of events.
• These events may arise from
  – the data (e.g. changing a value in a field)
  – the GUI (e.g. clicking on a button)
  – other SALT elements (e.g. miss-recognition triggers prompts).
Level of API

- VoiceXML uses forms and fields as its building blocks.
- This allows to bundle prompts and grammars together into fields that are executed one-by-one to reflect the turn-taking control model of telephone dialogs.
Level of API

- SALT applies a lower level interface
  - to offer finer grain manipulation of speech input and output.
- SALT leaves
  - the customisation of turn-talking and
  - dialog flow to the application author.
The Future

- Although VoiceXML and SALT were developed to solve different problems, these problem spaces are beginning to converge.
- Some VoiceXML developers are asking for a stripped-down version of VoiceXML without FIA so that they can write their own turn-taking strategies.
- Other VoiceXML developers are asking that VoiceXML be modularised so that its tags can be embedded into other languages.
Take-Home Messages

• SALT is an extension to HTML that enables developers to add a spoken dialog interface to Web applications.
• Using SALT, speech applications can be written for
  – voice-only browsers
  – multimodal browsers.
• Client runs SALT applications using IE and Speech Add-in.