SALT: A SPOKEN LANGUAGE INTERFACE FOR WEB-BASED MULTIMODAL DIALOG SYSTEMS

Kuansan Wang

Speech Technology Group, Microsoft Research
One Microsoft Way, Redmond, WA 98052 USA
http://research.microsoft.com/stg

ABSTRACT

This paper describes the Speech Application Language Tags, or SALT, an emerging spoken language interface standard for multimodal or speech-only applications. A key premise in SALT design is speech-enabled user interface shares a lot of the design principles and computational requirements with the graphical user interface (GUI). As a result, it is logical to introduce into speech the object-oriented, event-driven model that is known to be flexible and powerful enough in meeting the requirements for realizing sophisticated GUIs. It is hopeful that reusing this rich infrastructure can enable dialog designers to focus more on the core user interface design issues than on the computer and software engineering details. The paper centers the discussion on the Web-based distributed computing environment and elaborates how SALT can be used to implement multimodal dialog systems. How advanced dialog effects (e.g., cross-modality reference resolution, implicit confirmation, multimedia synchronization) can be realized in SALT is also discussed.

1. INTRODUCTION

Multimodal interface allows a human user to interaction with the computer using more than one input methods. GUI, for example, is multimodal because a user can interact with the computer using keyboard, stylus, or pointing devices. GUI is an immensely successful concept, notably demonstrated by the World Wide Web. Although the relevant technologies for the Internet had long existed, it was not until the adoption of GUI for the Web did we witness a surge on its usage and rapid improvements in Web applications.

GUI applications have to address the issues commonly encountered in a goal-oriented dialog system. In other words, GUI applications can be viewed as conducting a dialog with its user in an iconic language. For example, it is very common for an application and its human user to undergo many exchanges before a task is completed. The application therefore must manage the interaction history in order to properly infer user's intention. The interaction style is mostly system initiative because the user often has to follow the prescribed interaction flow where allowable branches are visualized in graphical icons. Many applications have introduced mixed initiative features such as type-in help or search box. However, user-initiated digressions are often recoverable only if they are anticipated by the application designers. The plan-based dialog theory [1][2][3] suggests that, in order for the mixed initiative dialog to function properly, the computer and the user should be collaborating partners that actively assist each other in planning the dialog flow. An application will be perceived as hard to use if the flow logic is obscure or unnatural to the user and, similarly, the user will feel frustrated if the methods to express intents are too limited. It is widely believed that spoken language can improve the user interface as it provides the user a natural and less restrictive way to express intents and receive feedbacks.

The Speech Application Language Tags (SALT) [4] is a proposed standard for implementing spoken language interfaces. The core of SALT is a collection of objects that enable a software program to listen, speak, and communicate with other components residing on the underlying platform (e.g., discourse manager, other input modalities, telephone interface, etc.). Like their predecessors in the Microsoft Speech Application Interface (SAPI), SALT objects are programming language independent. As a result, SALT objects can be embedded into a HTML or any XML document as the spoken language interface [5]. Introducing speech capabilities to the Web is not new [6][7][8]. However, it is the utmost design goal of SALT that advanced dialog management techniques (e.g., [9-12]) can be realized in a straightforward manner in SALT.

2. DIALOG ARCHITECTURE OVERVIEW

With the advent of XML Web services, the Web has quickly evolved into a gigantic distributed computer where Web services, communicating in XML, play the role of reusable software components. Using the universal description, discovery, and integration (UDDI) standard, Web services can be discovered and linked up dynamically to collaborate on a task. In other words, Web services can be regarded as the software agents envisioned in the open agent architecture [12]. Conceptually, the Web infrastructure provides a straightforward means to realize the agent-based approach suitable for modeling highly sophisticated dialog [1]. This distributed model shares the same basis as the SALT dialog management architecture.

2.1. Page-based Dialog Management

An examination on human to human conversation on trip planning shows that experienced agents often guide the customers in dividing the trip planning into a series of more manageable and relatively untangled subtasks [10]. Not only the observation contributes to the formation of the plan-based dialog theory, but the same principle is also widely adopted in designing GUI-based transactions where the subtasks are usually encapsulated in visual pages. Take a travel planning Web site for example. The first page usually gathers some basic

information of the trip, such as the traveling dates and the originating and destination cities, etc. All the possible travel plans are typically shown in another page, in which the user can negotiate on items such as the price, departure and arrival times, etc. To some extent, the user can alter the flow of interaction. If the user is more flexible for the flight than the hotel reservation, a well designed site will allow the user to digress and settle the hotel reservation before booking the flight. Necessary confirmations are usually conducted in separate pages before the transaction is executed.

The designers of SALT believe that spoken dialog can be modeled by the page-based interaction as well, with each page designed to achieve a sub-goal of the task. There seems no reason why the planning of the dialog cannot utilize the same mechanism that dynamically synthesizes the Web pages today.

2.2. Separation of Data and Presentation

SALT preserves the tremendous amount of flexibility of a page-based dialog system in dynamically adapting the style and presentation of a dialog [5]. A SALT page is composed of three portions: (1) a data section corresponding to the information the system needs to acquire from the user in order to achieve the sub-goal of the page; (2) a presentation section that, in addition to GUI objects, contains the templates to generate speech prompts and the rules to recognize and parse user's utterances; (3) a script section that includes inference logic for deriving the dialog flow in achieving the goal of the page. The script section also implements the necessary procedures to manipulate the presentation sections.

This document structure is motivated by the following considerations. First, the separation of the presentation from the resst localizes the natural language dependencies. An application can be ported to another language by changing only the presentation section without affecting other sections. Also, a good dialog must dynamically strike a balance between system initiative and user initiative styles. However, the needs to switch the interaction style do not necessitate changes in the dialog planning. The SALT document structure maintains this type of independence by separating the data section from the rest of the document, so that when there are needs to change the interaction style, the script and the presentation sections can be modified without affecting the data section. The same mechanism also enables the app to switch among various UI modes, such as in the mobile environments where the interactions must be able to seamlessly switching between a GUI and speech-only modes for hand-eye busy situations. The presentation section may vary significantly among the UI modes, but the rest of the document can remain largely intact.

2.3. Semantic Driven Multimodal Integration

SALT follows the common GUI practice and employs an object-oriented, event-driven model to integrate multiple input methods. The technique tracks user's actions and reports them as events. An object is instantiated for each event to describe the causes. For example, when a user clicks on a graphical icon, a mouse click event is fired. The mouse-click event object contains information such as coordinates where the click takes place. SALT extends the mechanism for speech input, in which the notion of semantic objects [5][13] is introduced to capture the meaning of spoken language. When the user says something, speech events, furnished with the corresponding semantic objects, are reported. The semantic objects are structured and categorized. For example, an utterance "Send mail to John" is composed of two nested semantic objects: "John" representing the semantic type "Person" and the whole utterance the semantic type "Email command." SALT therefore enables a multimodal integration algorithm based on semantic type compatibility [14]. The same command can be manifest in a multimodal expression, as in "Send email to him [click]" where the email recipient is given by a point-and-click gesture. Here the semantic type provides a straightforward way to resolve the cross modality reference: the handler for the GUI mouse click event can be programmed into producing a semantic object of the type "Person" which can subsequently be identified as a constituent of the "email command" semantic object.

Because the notion of semantic objects is quite generic, dialog designers should find little difficulty employing other multimodal integration algorithms, such as the unification based approach described in [15], in SALT.

3. BASIC SPEECH OBJECTS IN SALT

SALT speech objects encapsulate speech functionality. They resemble to the GUI objects in many ways. Because they share the same high level abstraction, SALT speech objects interoperate with GUI objects in a seamless and consistent manner. Multimodal dialog designers can elect to ignore the modality of communication, much the same way as they are insulated from having to distinguish whether a text string is entered to a field through a keyboard or cut and pasted with a pointing device.

3.1. The Listen Object

The "listen" object in SALT is the speech input object. The object must be initialized with a speech grammar that defines the language model and the lexicon relevant to the recognition task. The object has a *start* method that, upon invocation, collects the acoustic samples and performs speech recognition. If the language model is a probabilistic context free grammar (PCFG), the object can return the parse tree of the recognized outcome. Optionally, dialog designers can embed XSLT templates or scripts in the grammar to shape the parse tree into any desired format. The most common usage is to transform the parse tree into a semantic tree composed of semantic objects.

A SALT object is instantiated in an XML document whenever a tag bearing the object name is encountered. For example, a listen object can be instantiated as follows:

```
<listen id="foo" onreco="f()" onnoreco="g()"  mode="automatic">
   <grammar name="main" src="../meeting.xml"/>
</listen>
```

The object, named "foo," is given a speech grammar whose universal resource indicator (URI) is specified via a <grammar> constituent. As in the case of HTML, methods of an object are invoked via the object name. For example, the command to start the recognition is foo.start() in the ECMAScript syntax. Upon a successful recognition and parsing, the listen object raises the event "onreco." The event handler, f(), is associated in the HTML syntax as shown above. If the recognition result is rejected, the listen object raises the "onnoreco" event, which, in

the above example, invokes function g(). As mentioned in Sec. 2.2, these event handlers reside in the script section of a SALT page that manages the within-page dialog flow. Note that SALT is designed to be agnostic to the syntax of the eventing mechanism. Although the examples through out this article use HTML syntax, SALT can operate with other eventing standards, such as World Wide Web Consortium (W3C) XML Document Object Model (DOM) Level 2, ECMA Common Language Infrastructure (CLI), or the upcoming W3C proposal called XML Events.

The SALT listen object can operate in one of the three modes designed to meet different UI requirements. The *automatic* mode, shown above, automatically detects the end of utterance and cut off the audio stream. The mode is most suitable for push-to-talk UI or telephony based systems. Reciprocal to the *start* method, the listen object also has a *stop* method for forcing the recognizer to stop listening. The designer can explicitly invoke the stop method and not rely on the recognizer's default behavior. Invoking the stop method becomes necessary when the listen object operates under the *single* mode, where the recognizer is mandated to continue listening until the stop method is called. Under the single mode, the recognizer is required to evaluate and return hypotheses based on the full length of the audio, even though some search paths may have reached a legitimate end of sentence token in the middle of the audio stream. In contrast, the third *multiple* mode allows the listen object to report hypotheses as soon as it sees fit. The single mode is designed for push-hold-and-talk type of UI, while the multiple mode is for real-time or dictation type of applications.

The listen object also has methods to modify the PCFG it contains. Rules can be dynamically activated and deactivated to control the perplexity of the language model. The semantic parsing templates in the grammar can be manipulated to perform simple reference resolution. For example, the grammar below (in SAPI format) demonstrates how a deictic reference can be resolved inside the SALT listen object:

```
<rule propname="drink" …>
  <option> the </option>
  <list> <phrase propvalue="coffee"> left </phrase>
        <phrase propvalue="juice"> right </phrase> </list>
  <option> one </option>
</rule>
```

In this example, the *propname* and *propvalue* attributes are used to generate the semantic objects. If the user says "the left one," the above grammar directs the listen object to return the semantic object as <drink text="the left one">coffee</drink>. This mechanism for composing semantic objects is particularly useful for processing expressions closely tied to how data are presented. The grammar above may be used when the computer asks the user for choice of the drink by displaying the pictures of the choices side by side. However, if the display is tiny, the choices may be rendered as a list, to which a user may say "the first one" or "the bottom one." SALT allows dialog designers to approach this problem by dynamically adjusting the speech grammar.

3.2. The prompt object

The SALT "prompt" object is the speech output object. Like the listen object, the prompt object has a *start* method to begin the audio playback. The prompt object can perform text to speech synthesis (TTS) or play pre-recorded audio. For TTS, the prosody and other dialog effects can be controlled by marking up the text with synthesis directives.

Barge-in and bookmark are two events of the prompt object particularly useful for dialog designs. The prompt object raises a barge-in event when the computer detects user utterance during a prompt playback. SALT provides a rich program interface for the dialog designers to specify the appropriate behaviors when the barge-in occurs. Designers can choose whether to delegate SALT to cut off the outgoing audio stream as soon as speech is detected. Delegated cut-off minimizes the barge-in response time, and is close to the expected behavior for users who wish to expedite the progress of the dialog without waiting for the prompt to end. Similarly, non-delegated barge-in let the user change playback parameters without interrupting the output. For example, the user can adjust the speed and volume using speech commands while the audio playback is in progress. SALT will automatically turn on echo cancellation for this case so that the playback has minimal impacts on the recognition.

The timing of certain user action or the lack thereof often bears semantic implications. Implicit confirmation is a good example, where the absence of an explicit correction from the user is considered as a confirmation. The prompt object introduces an event for reporting the landmarks of the playback. The typical way of catching the playback landmarks in SALT is as such:

```
<prompt id="bar" onbookmark="f()" …>
   Traveling to New York? <bookmark name="imp_confirm"/>
   There are <emph> 3 </emph> available flights …
</prompt>
```

When the synthesizer reaches the TTS markup <bookmark>, the onbookmark event is raised and the event hander f() is invoked. When a barge-in is detected, the dialog designer can determine if the barge-in occurs before or after the bookmark by inspecting whether the function f() has been called or not.

Multimedia synchronization is another main usage for TTS bookmarks. When the speech output is accompanied with, for example, graphical animations, TTS bookmarks are an effective mechanism to synchronize these parallel outputs.

To include dynamic content in the prompt, SALT adopts a simple template-based approach for prompt generation. In other words, the carrier phrases can be either pre-recorded or hard-coded, while the key phrases can be inserted and synthesized dynamically. The prompt object that confirms a travel plan may appear as the following in HTML:

```
<input name="origin" type="text" />
<input name="destination" type="text" />
<input name="date" type="text" />
…
<prompt …> Do you want to fly from
   <value targetElement="origin"/> to
   <value targetElement="destination"/> on
   <value targetElement="date"/>?
</prompt>
```

As shown above, SALT uses a <value> tag inside a prompt object to refer to the data contained in other parts of the SALT page. In this example, the prompt object will insert the values in the HTML input objects in synthesizing the prompt.

3.3. Declarative Rule-based Programming

Although the examples use procedural programming in managing the dialog flow control, SALT designers can practice inference programming in a declarative rule-based fashion in which rules are attached to the SALT objects capturing user's actions, e.g., the listen object. Instead of authoring procedural event handlers, designers can declare inside the listen object rules that will be evaluated and invoked when the semantic objects are returned. This is achieved through a SALT <bind> element as demonstrated below:

```
<listen ...>  <grammar .../>
    <bind test="/@confidence $lt$ 50"
          targetElement="prompt_confirm" targetMethod="start"
          targetElement="listen_confirm" targetMethod="start" />
    <bind test="/@confidence $ge$ 50"
          targetElement="origin" value="/city/origin"
          targetElement="destination" value="/city/destination"
          targetElement="date" value="/date" />     ...
</listen>
```

The predicate of each rule is applied in turns against the result of the listen object. They are expressed in the standard XML Pattern language in the "test" clause of the <bind> element. In this example, the first rule checks if the confidence level is above the threshold. If not, the rule activates a prompt object (*prompt_confirm*) for explicit confirmation, followed by a listen object *listen_confirm* to capture the user's response. The speech objects are activated via the *start* method of the respective object. Object activations are specified in the *targetElement* and the *targetMethod* clauses of the <bind> element. Similarly, the second rule applies when the confidence score exceeds the prescribed level. The rule extracts the relevant semantic objects from the parsed outcome and assigns them to the respective elements in the SALT page. As shown above, SALT reuses the W3C XPATH language for extracting partial semantic objects from the parsed outcome.

4. SUMMARY

This paper describes how SALT can be employed to realize multimodal dialog systems using the Web architecture. With the observation that Web based applications generally practice the same UI principles necessary to design spoken dialog, SALT is designed to empower dialog designers to leverage the rich infrastructure of the Web. Accordingly, SALT embraces the object-oriented, event-driven programming techniques that are already familiar to the Web programmers.

The basic speech objects in SALT are the listen and prompt objects for speech input and output, respectively. We elaborate how advanced dialog effects can be achieved by using these two objects. For the listen object, we show how to resolve cross-modality anaphoric and deictic references, and how to adapt reference resolution dynamically to the UI presentation. We also show how implicit confirmation and multimedia synchronization can be achieved using the SALT prompt object.

SALT undoubtedly needs further improvement. The most obvious omission in its current form is the lack of natural language generation and pragmatic programming. Although one can introduce the notion of speech acts and pragmatics into SALT documents, there is no native support and standard way to do so. These are the areas where relevant advancements in computational linguistics can be used to improve SALT as it is today.

5. References

[1] Sadek, M.D., Bretier, P., Panaget F., "ARTIMIS: Natural dialog meets rational agency," Proc. IJCAI-97, Japan, 1997.

[2] Allen, J.F., Natural Language Understanding, 2nd Ed., Benjamin-Cummings, Redwood City, CA, 1995.

[3] Cohen, P.R, Morgan, J., Pollack, M.E., Intentions in Communications, MIT Press, Cambridge MA, 1990.

[4] Speech and Language Tags (SALT) Forum, http://www.saltforum.org.

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

[6] Aron, B., "Hyperspeech: Navigating in speech-only hypermedia," Proc. Hypertext 91, San Antonio, TX, 1991.

[7] Ly, E., Schmandt, C., Arons, B., "Speech recognition architectures for multimedia environments," Proc. AVIOS-93, San Jose, CA., 1993.

[8] Lau, R., Flammia, G., Pao, C., Zue, V., "Webgalaxy: Integrating spoken language and hyptertext navigation," Proc. Eurospeech-97, Rhodes, Greece, 1997.

[9] Sneff, S., Hurley, E., Lau, R., Pao, C., Schmid, P., Zue, V., "Galaxy-II: A reference architecture for conversational system development," Proc. ICSLP-98, Sydney, Australia, 1998.

[10] Rudnicky, A., Xu, W., "An agenda-based dialog management architecture for spoken language systems," Proc. ASRU-99, Keystone, Co., 1999.

[11] Lin, B.-S, Wang, H.-M, Lee, L.-S., "A distributed architecture for cooperative spoken dialog agents with coherent dialog state and history," Proc. ASRU-99, Keystone, Co., 1999.

[12] Bradshaw, J.M. (Ed), Software Agents, AAAI/MIT Press, Cambridge, MA, 1996.

[13] Wang, K., "An event driven model for dialog systems," Proc. ICSLP-98, Sydney, Australia, 1998.

[14] Wang, K., "Semantic modeling for dialog systems in a pattern recognition framework," Proc. ASRU-2001, Trento, Italy, 2001.

[15] Johnston, M., Cohen, P.R., McGee, D., Oviatt, S.L., Pittman, J.A., Smith, I., "Unification based multimodal integration," Proc. 35th ACL, Madrid, Spain, 1997.