

to go to Seattle?”“Yes, to Seattle”), to correct by saying “no”+new value (e.g., “Do you want to go to Seattle?”“No, Pittsburgh”).

[0106] Statement Control

[0107] The statement control allows the application developer to provide an output upon execution of the client side markup when a response is not required from the user of the client device 30. An example could be a “Welcome” prompt played at the beginning of execution of a client side markup page.

[0108] An attribute can be provided in the statement control to distinguish different types of information to be provided to the user of the client device. For instance, attributes can be provided to denote a warning message or a help message. These types could have different built-in properties such as different voices. If desired, different forms of statement controls can be provided, i.e. a help control, warning control, etc. Whether provided as separate controls or attributes of the statement control, the different types of statements have different roles in the dialog created, but share the fundamental role of providing information to the user of the client device without expecting an answer back.

[0109] Eventing

[0110] Event handlers as indicated in FIG. 11 are provided in the QA control 320, the output controls 308 and the input controls 310 for actions/inactions of the user of the client device 30 and for operation of the recognition server 204 to name a few, other events are specified in Appendix B. For instance, mumbling, where the speech recognizer detects that the user has spoken but is unable to recognize the words and silence, where speech is not detected at all, are specified in the QA control 320. These events reference client-side script functions defined by the author. In a multimodal application specified earlier, a simple mumble handler that puts an error message in the text box could be written as follows:

```

<Speech:QA
  controlsToSpeechEnable="txtDepCit
  y" onClientNoReco="OnMumble ( )"
  runat="server" >
  <Answer id="AnsDepCity"
    StartEvent="onMouseDown"
    StopEvent="onMouseUp"
    >
    <grammar src="/grammars/depCities.gram"/>
    <bind value="//sml/DepCity"
targetElement="txtCity" />
  </Answer>
</Speech:QA>
<script>
  function OnMumble ( ) {
    txtDepCity.value=". . . recognition
error . . .";
  }
</script>

```

[0111] Control Execution Algorithm

[0112] In one embodiment, a client-side script or module (herein referred to as “RunSpeech”) is provided to the client device. The purpose of this script is to execute dialog flow via logic, which is specified in the script when executed on

the client device 30, i.e. when the markup pertaining to the controls is activated for execution on the client due to values contained therein. The script allows multiple dialog turns between page requests, and therefore, is particularly helpful for control of voice-only dialogs such as through telephony browser 216. The client-side script RunSpeech is executed in a loop manner on the client device 30 until a completed form is submitted, or a new page is otherwise requested from the client device 30.

[0113] It should be noted that in one embodiment, the controls can activate each other (e.g. question control activating a selected answer control) due to values when executed on the client. However, in a further embodiment, the controls can “activate” each other in order to generate appropriate markup, in which case server-side processing may be implemented.

[0114] Generally, in one embodiment, the algorithm generates a dialog turn by outputting speech and recognizing user input. The overall logic of the algorithm is as follows for a voice-only scenario:

- [0115] 1. Find next active output companion control;
- [0116] 2. If it is a statement, play the statement and go back to 1; If it is a question or a confirm go to 3;
- [0117] 3. Collect expected answers;
- [0118] 4. Collect commands;
- [0119] 5. Play output control and listen in for input;
- [0120] 6. Activate recognized Answer or Command object or, issue an event if none is recognized;
- [0121] 7. Go back to 1.

[0122] In the multimodal case, the logic is simplified to the following algorithm:

- [0123] 1. Wait for triggering event—i.e., user tapping on a control;
- [0124] 2. Collect expected answers;
- [0125] 3. Listen in for input;
- [0126] 4. Activate recognized Answer object or, if none, throw event;
- [0127] 5. Go back to 1.

[0128] The algorithm is relatively simple because, as noted above, controls contain built-in information about when they can be activated. The algorithm also makes use of the role of the controls in the dialogue. For example statements are played immediately, while questions and confirmations are only played once the expected answers have been collected.

[0129] In a further embodiment, implicit confirmation can be provided whereby the system confirms a piece of information and asks a question at the same time. For example the system could confirm the arrival city of a flight and ask for the travel date in one utterance: “When do you want to go to Seattle?” (i.e. asking ‘when’ and implicitly confirming ‘destination: Seattle’). If the user gives a date then the city is considered implicitly accepted since, if the city was wrong, users would have immediately challenged it. In this scenario, it becomes clear that the knowledge of what a user is trying to achieve is vitally important: are they answering