Architectural Design – Interactors

1. Single List Selection

Read the list of items and be ready to accept a response (yes, yeah, that one) after each item. Only one item from the list can be selected.

Input: String initial prompt, String comma-separated list of options, id of output control
Output: String representing user choice
Action:

1. Read the initial prompt
2. Read the list of options, allowing user to interrupt
3. If user interrupts with "yes', "yeah", or "that one", interrupt, return the name of the most recently-read choice, and increment the 'Match' counter
4. If user interrupts with another response, continue reading and increment the 'No Match' counter
5. If no response is received, increment the 'no response' counter
6. If no valid response is received, return null

e.g. Prompt: "Would you like to buy apples, oranges, bananas?" The user: "That one" after hearing oranges".

2. Multiple List Selection (using speech)

Read the list of items and be ready to accept a response (yes, yeah, that one) after each item. After user selects an item, prompt to see if he/she wants to select other item. If yes, then read the rest of the list listening for the response. Repeat after each selected item until the end of the list is reached. One or more items can be selected.

Input: String initial prompt, String comma-separated list of options, id of output control, string intermediate prompt, optional String continuation prompt
Output: String representing user choices, separated by commas
Action:

1. Read the initial prompt
2. Read the list of options, allowing user to interrupt
3. If user interrupts with "yes', "yeah", or "that one", interrupt playback, increment the 'match' counter, and return the name of the most recently-read choice
    i. Add the selection to the output string
    ii. Prompt user with "you selected", followed by the text of their choice
    iii. If additional choices remain in the list, do the following:
        1. If a continuation prompt was provided, read the continuation prompt
        2. Otherwise, prompt with "Would you like to make an additional selection?"

3. If the user answers “yes,” or “yeah,” read the intermediate prompt, increment the ‘match’ counter and go back to step 2, starting from the next item in the list
4. If the user answers “no,” increment the ‘match’ counter
5. If the user response is not understood, increment the ‘no match’ counter
6. If the user does not response, increment the ‘no response’ counter
7. Return the output string

4. If user interrupts with another response, increment the ‘no match’ counter and continue reading
5. If no response is given, increment the ‘no response’ counter; if no valid response is given, return null

*e.*g. Prompt: “Would you like to buy apples, ..?” User says, “yes” after hearing apples. Computer response: “You selected apples. Would you like to buy something else?” User says “yes”. Prompt: “oranges, bananas”. User says “yeah” after hearing bananas. Computer response: “You selected bananas”. (The end of the list is reached)

3. Multiple List Selection (using speech) With Deselecting

Read the list of items and be ready to accept a response (yes, yeah, that one) after each item. After user selects an item, prompt to see if he/she wants to select other item. If yes, then read the rest of the list listening for the response. Repeat after each selected item until the end of the list is reached. When user is done selecting, read the list of selected items back to him/her, and instruct the user to say “no” after the item they would like to deselect. One or more items can be selected or deselected.

Input: String initial prompt, String comma-separated list of options, id of output control, string intermediate prompt, optional String continuation prompt, optional string verification prompt
Output: String representing user choices, separated by commas
Action:

1. Read the initial prompt
2. Read the list of options, allowing user to interrupt
3. If user interrupts with “yes’, “yeah”, or “that one”, interrupt playback, increment the ‘match’ counter, and return the name of the most recently-read choice
    i. Add the selection to the output list
    ii. Prompt user with “you selected”, followed by the text of their choice
    iii. If additional choices remain in the list, do the following:
        1. If a continuation prompt was provided, read the continuation prompt
        2. Otherwise, prompt with “Would you like to make an additional selection?”
        3. If the user answers “yes’ or “yeah,” increment the ‘match’ counter, read the intermediate prompt and go back to step 2, starting from the next item in the list

4. If the user answers "no," increment the 'match' counter
5. If the user answer is not understood, increment the 'no match' counter
6. If the user does not respond, increment the 'no response' counter
7. Go to step 6

4. If user interrupts with another response, increment the 'no match' counter and continue reading
5. If no response is given, increment the 'no response' counter. If no valid response was given, return null
6. If a verification prompt was provided, read the verification prompt, otherwise read "You made the following selections – you may remove any selection by saying 'no' after the selection is read."
7. Read back each item in the output list, allowing the user to interrupt
8. If the user interrupts with "no", increment the 'match' counter and go back to step 7, starting at the next item.
9. If the user interrupts with something else, increment the 'no match' counter, add the item to the output list, and go back to step 7, starting from the next item
10. Otherwise add the item to the output list and go back to step 7, starting from the next item
11. Return the output list

e.g. Prompt: "Would you like to buy apples, ..?" User says, "yes" after hearing apples. Computer response: "You selected apples. Would you like to buy something else?" User says "yes". Prompt: "oranges, bananas". User says "yeah" after hearing bananas. Computer response: "You selected bananas". (The end of the list is reached) Prompt: "Here are the items you selected. Say no after an item you don't want to keep. Apples, Oranges". User says no after bananas. (Bananas are taken off the list)

4. Double Yes/No (using speech)

When asked yes/no question, the user says something else (yeah, may be, neh …) Will confirm unclear response by asking "Did you say Yes/No?" Will be used on error (user says something but yes/no) or on low confidence level (user says something close to yes/no). Repeat the confirmation only once.

Input: String initial prompt, id of output control, optional String 'yes' prompt, optional string 'no' prompt
Output: String representing user choice
Action:

1. Read the initial prompt, increment the 'iterations' count and wait for a response
2. If "yes" or "yeah" is chosen with low confidence, increment the 'match' counter
    i. Read the "yes" prompt if one was provided
    ii. Otherwise ask, "Was that a 'yes'?"
    iii. Wait for response
    ii. If no, return "no", increment the '2nd level no,' and 'match' counters
    iii. If "yes", return "yes", increment the 'match' counter

iv. If the user response is not understood, increment the 'no match' counter, If the number of iterations < 3, go to step 1, otherwise return 'yes'
v. If no response, increment the 'no response' counter and return "yes"

3. If "no" is chosen with low confidence, increment the 'match' counter
   i. Read the "no" prompt if one was provided
   ii. Otherwise ask, "Was that a 'no'?"
   iii. Wait for response
   iv. If no, return "yes," increment the '2nd level no' and 'match' counters
   v. If "yes", return "no," increment the 'match' counter
   vi. If the user response is not understood, increment the 'no match' counter. If the number of iterations < 3, go to step 1, otherwise return 'no'
   vii. If no response, increment the 'no response' counter and return "no"
4. If "yes" or "yeah" is chosen with high confidence, return "yes", increment the 'match' counter
5. If "no" is chosen with high confidence, return "no", increment the 'match' counter
6. If the response does not match the grammar, increment the 'no match' counter and return null
7. If no response is given, increment the 'no response' counter and return null

Note: ask user 10 yes/no questions and see how many times an error or low confidence level occurs. This will give an idea if this interactor is actually useful.

e.g. Prompt: "Would you like a single room?" User says "Ya". Computer response: "Did you say yes?" User says, "Yes". (The answer to the original question was yes) OR In response to "did you say yes" user says "No". (The answer to the original question was no)

5. Double Yes/No (using DTMF)

When asked yes/no question, the user says something else (yeah, may be, neh …) Will confirm unclear response by saying "If you said yes, press 1, if you said no, press 2". Will be used on error (user says something but yes/no) or on low confidence level (user says something close to yes/no). Repeat the confirmation only once.

Note: ask user 10 yes/no questions and see how many times an error or low confidence level occurs. This will give an idea if this interactor is actually useful.

Characteristics: will be clearer than Double Yes/No using Speech, but does it make sense to use?

Input: String initial prompt, id of output control, optional String 'yes' prompt, optional string 'no' prompt
Output: String representing user choice

Action:

1. Read the initial prompt and wait for a response
2. If "yes" is chosen with low confidence, increment the 'match' counter
    i. Read the "yes" prompt if one was provided
    ii. Otherwise ask "press one if your answer was 'yes', press 2 if your answer was 'no'"
    iii. Wait for response
    iv. If 1, return "yes"
    v. If 2, return "no"
    vi. If no response, increment the 'no response' counter and return "yes"
3. If "no" is chosen with low confidence, increment the 'match' counter
    i. Read the "no" prompt if one was provided
    ii. Otherwise ask "press one if your answer was 'yes', press 2 if your answer was 'no'"
    iii. Wait for response
    iv. If 1, return "yes"
    v. If 2, return "no"
    vi. If no response, increment the 'no response' counter and return "no"
4. If "yes" or "yeah" is chosen with high confidence, increment the 'match' counter and return "yes."
5. If "no" is chosen with high confidence, increment the 'match' counter and return "no."
6. If the user response does not match the grammar, increment the 'no match' counter and return null
7. If no response is given, increment the 'no response' counter and return null

e.g. Prompt: "Would you like a single room?" User says "Ya". Computer response: "If you said yes, press 1, if you said no, press 2?" User presses 1. (The answer to the original question was yes) OR User presses 2. (The answer to the original question was no)

6. Select From N-Best List

Confirm an answer to a "textual" question using a proximity threshold in the n-best list. After user's response, get the list of words that match that response the most, and prompt the user again with that list of words: "Did you say item1, item2 or item3?"

Input: String initial prompt, String comma-separated list of options, id of output control, optional real number threshold
Output: String representing user choice
Action:

1. Read the initial prompt, wait for user response
2. Examine returned n-best list
3. If one or more options is within threshold of top choice, increment the 'match' counter
    i. Read each item, allowing user to interrupt

ii. If user interrupts with "yes" or "yeah," or pushes "1," increment the 'match' counter and return the text of the last value read
iii. If user interrupts with something else, increment the 'no match' counter and continue reading from the list
iv. If user does not respond, increment the 'no response' counter and return the top choice

4. Otherwise, increment the 'match' counter and return the selection made

e.g. "What city would you like to visit?" User says "Long View". Computer response: "Did you say Long View, Border Butte, or Fond Duke?" User says yes after hearing Long View. (The city user wanted to visit was Long View)

7. Different Prompts for Different Responses (n-ary prompt)

If user does not respond, prompt in a certain way. If user's response does not match the grammar, prompt in a different way.

Input: String initial prompt, id of output control, string help prompt, string "no response" prompt, string "invalid response" prompt, string SRGS grammar of choices
Output: String representing user choice
Action:

1. Read the initial prompt
2. Wait for user response, increment the 'iterations' counter
3. If response is valid, increment the 'match' counter and return the response
4. If user responds with "help", increment the 'match' counter, read the help prompt, and go back to step 2
5. If response is an error, increment the 'no match' counter. If iterations < 3 read the "invalid response" prompt and go to step 2, otherwise return null
6. If no response, increment the 'no response'. If iterations < 3 read the "no response" prompt and go to step 2, otherwise return null

8. Context-sensitive Help Prompt

Provide help to user based on type of incorrect response.

Input: String initial prompt, id of output control, string SRGS grammar of choices, string formatted list of responses, string formatted list of prompts, string 'no match' prompt, string 'no response' prompt
Output: String representing user choice
Action:

1. Read the initial prompt
2. Wait for user response, increment the 'iterations' counter
3. If response is valid, increment the 'match' counter

   a. If response matches the list of expected invalid responses, increment the 'help match' counter. If iterations < 3, read the indicated prompt and go to step 2; otherwise return null
   b. If response does not match the invalid responses, return the response (valid response)
4. If the response is not recognized, increment the 'no match' counter. If iterations < 3, read the 'no match' prompt and go to step 2. Otherwise return null.
5. If no response, increment the 'no response' counter. If iterations < 3 read the "no response" prompt and go to step 2, otherwise return null

9. Whisper prompt

Prompt the user with sample answer at a lower volume.

Input: String initial prompt, id of output control, string whisper prompt, optional string list of choices
Output: String representing user choice
Action:
1. Read the initial prompt
2. Pause
3. Lower volume and read whisper prompt
4. Wait for response
5. If response is valid, increment the 'match' counter and return the response
6. If response is not understood, increment the 'no match' counter and return null
7. If no response is given, increment the 'no response' counter and return null

e.g. Prompt: "What city would you like to visit". Prompt at lower volume: "Portland Oregon, Portland Maine, or Seattle Washington".

10. Confirmation and Correction Dialog after all selections.

Confirm the responses given in an html form. Read back each response in the html form and allow the user to say 'no' to correct a response – when the user says no, the voice interactor corresponding to that field is activated to get a new response. When the correction is finished, continue confirming with the next html form element.

Input: string initial prompt, string structured list of control ids to check, string structured list of spoken field names,
Output: none
Action:
1. Read the initial prompt – it should include something like "say 'no' after any incorrect response"
2. Say "you entered" then the value for each control in the list, followed by "for" and the name of the control, allowing interrupts

3. If the user interrupts with "no," increment the 'match' counter
    i. Activate the 'onClick' method of the associated html control
    ii. Go to step 2, starting from the next item in the list
4. If the user interrupts with something else, increment the 'no match' counter and continue at step 2 with the next item

11. Volume Adjuster

Prompt the user for the correct volume, speaking rate, and timeout adjuster – reuse these rates for each interactor on the page. Prompt the user to say "up, higher, or more" to increase volume or "low, lower, down" to decrease volume while a passage is being read, and "stop" when the volume is at the correct level. Repeat the prompts for speaking rate and timeout interval.

Input: string volume prompt, string rate prompt, string timeout prompt
Output: none
Action:

1. Begin reading the initial prompt – it should start with something like "say 'up', 'more', or 'higher' to increase the volume, or 'down', 'lower', or 'less' to decrease the volume, say 'ok', 'good', 'stop', 'done', or 'finished' when the volume is OK", followed by 10-20 seconds or so of text, allow the user to interrupt
2. If the user interrupts with 'up', 'more', or 'higher', increment the 'match' counter and increase the volume
3. If the user interrupts with 'down', 'lower', or 'less', increment the 'match' counter and decrease the volume
4. If the user interrupts with 'ok', 'good', 'stop', 'done', or 'finished', increment the 'match' counter, set the volume as appropriate, and return
5. Re-read the prompt
6. Repeat for speaking rate and timeout

e.g. Prompt: "Please say 'up' to increase the volume and 'down' to decrease the volume during the following passage. Say 'stop' when the volume is satisfactory" …

12. Multiple Related Field Selection

Prompt for a set of related fields, using semantic functions to disambiguate responses. The programmer specifies a set of primary fields, which must be filled in, a grammar for these fields, and a set of secondary fields and grammars, which have default values. When the user responds, fill in all possible fields and prompt for the fields that are left unfilled. Once all fields are filled, apply semantic functions to create a list of potential responses – read back each possible response and ask the user to say 'yes' after the correct one.

Input: string initial prompt, string structured list of primary fields, string structured list of SRGS grammar references for primary fields, string structured list of prompts for primary fields, string structured list of secondary fields, string, structured list of SRGS grammar references for secondary fields, string list of defaults for secondary fields, string name of semantic disambiguating function, string clarification prompt

Output: string structured list of field values

Action:

1. Read the initial prompt
2. Wait for responses
3. If a response matches the grammar for a particular field, increment the 'match' counter and fill in the response
4. If a response does not match any grammar, increment the 'no match' counter
5. For each secondary field that remains unfilled, set the field value to the given default and increment the 'no response' counter
6. For each primary field that remains unfilled, increment the 'no response' counter, read the associated prompt and wait for a response
    a. If the response is understood, increment the 'match' counter, set the field value to the returned response, and continue with the next field
    b. If the response is not understood, increment the 'no match' counter and set the field value to null
    c. If no response is given, increment the 'no response' counter and set the field value to null
7. Call the given semantic function – input is a structured list of field values, output is a structured list of prompts and the associated field values
8. If only a single item is returned, set the field values as indicated and return
9. If no valid output is provided, set all fields to null and return
10. Read the clarification prompt, or say "Did you mean?" if none was provided
11. For each entry in the output list, read the associated prompt and wait for a response
    a. If the user responds "yes," "yeah," or "that one," increment the 'match' counter, set the field values as indicated, and return
    b. If the user responds "no," increment the 'match' counter
    c. If the user responds with something else, increment the 'no match' counter
12. If the user did not respond, increment the 'no response' counter. If no valid response is returned, set all field values to 'null'

e.g. Prompt: "What date would you like to arrive?". User: "eleven, twelve, 2004". Prompt: "Did you mean December 11th 2004?" User says yes and interactor returns 12/11/2004.