
ORACLE®

Oracle Digital Assistant

The Complete Training

Design Practice: Unlocking Users

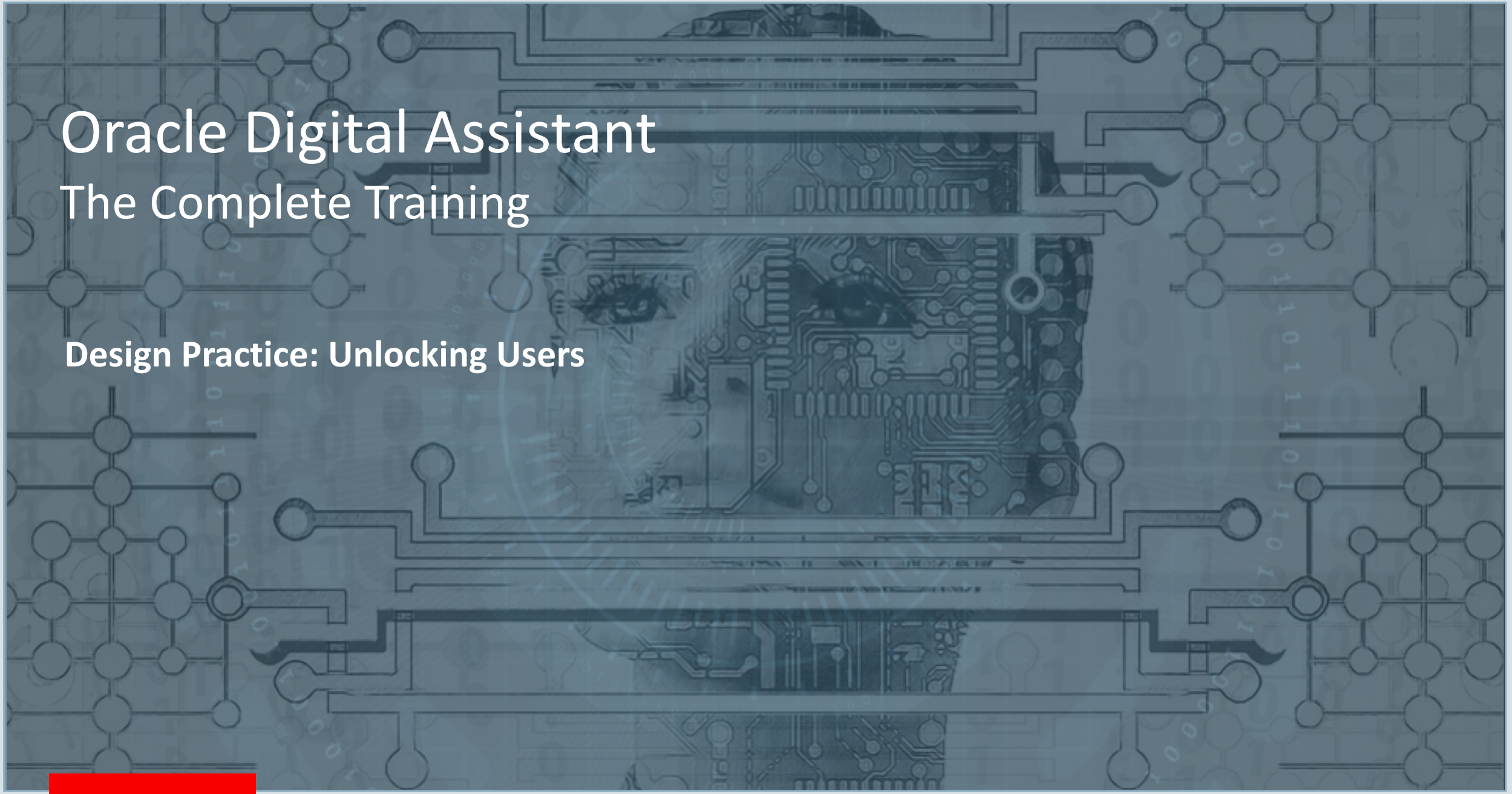


Image courtesy of pixabay.com

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

a bot is not a prison

help users to escape

Image courtesy of pixabay.com

Know what you want

- Good conversational design keeps users unlocked
  - Know exactly what you want for each state in a user interaction
  - Understand the consequences and the house keeping needed when a user interrupts or exits a conversation
  - Identify the best visual representation for each interaction in a conversation
  - Put your engineering mindset aside and try to think like a user
- Choose the best component for rendering the bot response
  - Common Response component allows you to build composite bot responses

The **tips** presented in this session **work with most input components**, not just those shown in the code examples

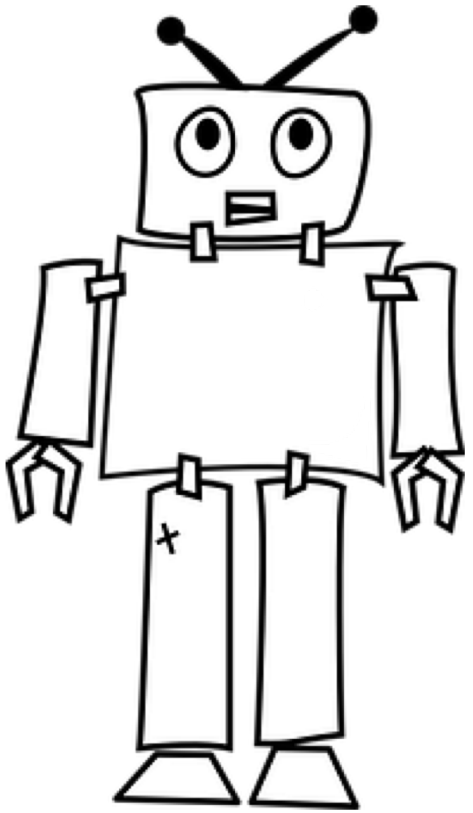


Image courtesy of pixabay.com

Topic agenda

1. Escaping the validation loop
2. Assist users with prompts
3. Show visual aids
4. Handle free text input
5. Use data input forms
6. Handle out-of-order messages

Topic agenda

1. Escaping the validation loop
2. Assist users with prompts
3. Show visual aids
4. Handle free text input
5. Use data input forms
6. Handle out-of-order messages

Referencing an entity type variable from an input component ensures **entity slotting**. However, this also **has side effects**.

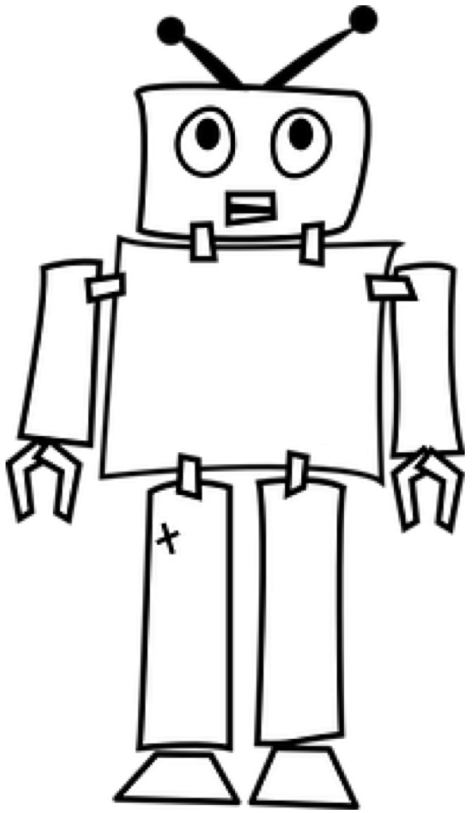
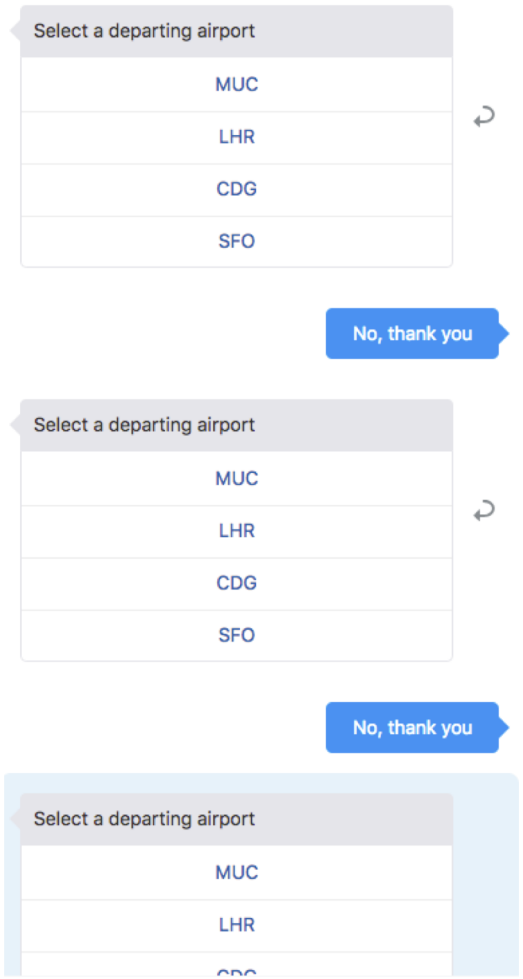


Image courtesy of pixabay.com

The validation loop

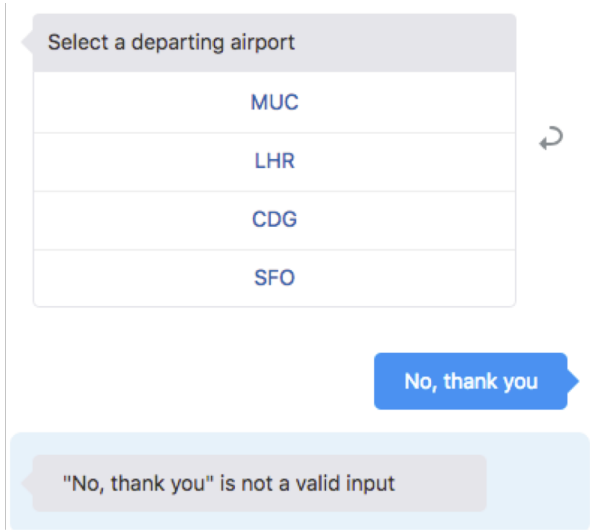



```
context:
  variables:
    airports: "AIRPORTS"
    iResult: "nlpresult"

states:
  selectAirport:
    component: "System.List"
    properties:
      prompt: "Select a departing airport"
      options: "${airports.type.enumValues}"
      variable: "airports"
      nlpResultVariable: "iResult"
    transitions:
      next: "printSelectedAirport"
```

- Components that reference entity type variables validate user input
- By default users are kept in the validation loop until a valid data entry is provided
- maxPrompts property allows you to define an exit

Exit the validation loop using maxPrompts property




```
selectAirport:
  component: "System.List"
  properties:
    prompt: "Select a departing airport"
    options: "${airports.type.enumValues}"
    variable: "airports"
    nlpResultVariable: "iResult"
    maxPrompts: 1
  transitions:
    next: "printSelectedAirport"
    actions:
      cancel: "handleMaxPromptsExceed"
```

```
handleMaxPromptsExceed:
  component: "System.Output"
  properties:
    text: "\"${system.message.messagePayload.text}\""
  transitions:
    return: "done"
```

- maxPrompts property can be set to any number > 0
  - Setting it to 2 means that users have 2 attempts for providing a correct value.
- Cancel action transition is followed when maximum number of failed input attempts are exceeded
- Handle cancel action
  - E.g. print message or route transition to intent state

The **entity type variable** associated with a component **is not updated** with user input when the number of maximum prompts is exceeded.

Use `${system.message.messagePayload.text}` to access the user input.

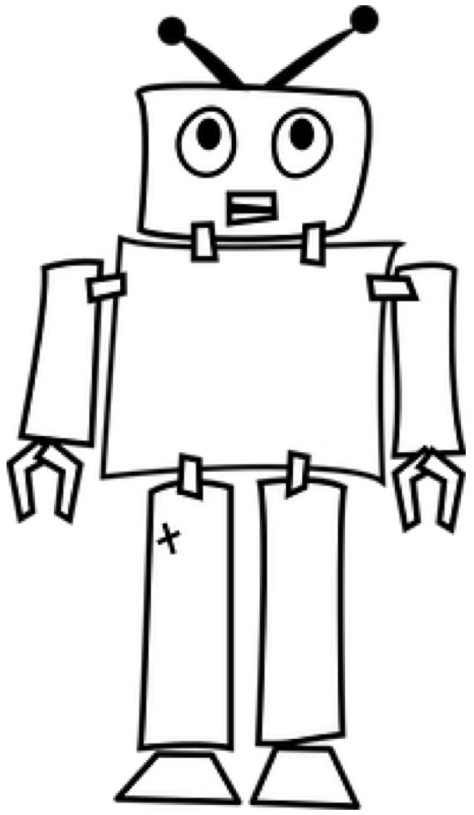
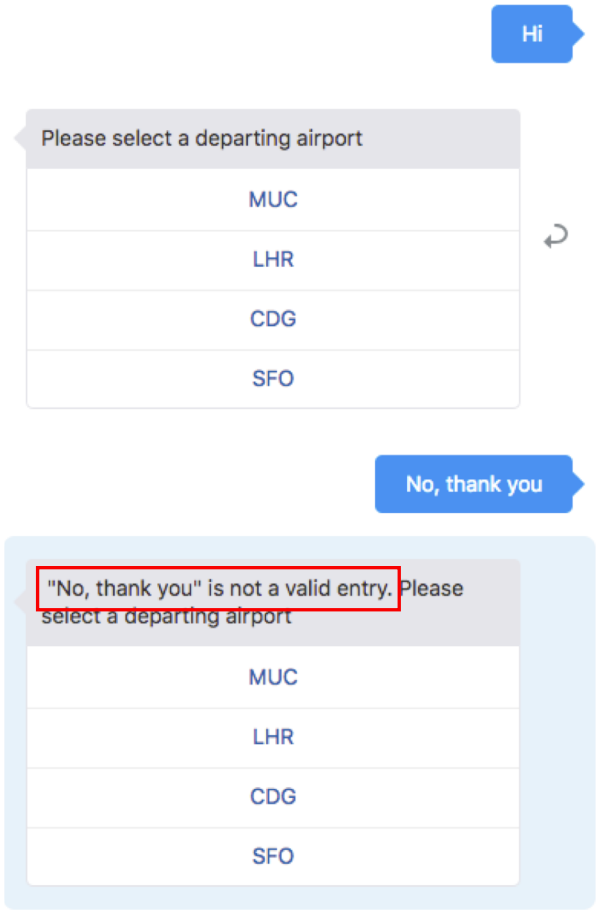


Image courtesy of pixabay.com

Topic agenda

1. Escaping the validation loop
2. Assist users with prompts
3. Show visual aids
4. Handle free text input
5. Use data input forms
6. Handle out-of-order messages

Improving the user experience




```
selectAirport:
  component: "System.List"
  properties:
    prompt: "<#if system.invalidUserInput?boolean>
             \"${system.message.messagePayload.text}\"
             is not a valid entry. </#if>Please
             select a departing airport"
    options: "${airports.type.enumValues}"
    variable: "airports"
    nlpResultVariable: "iResult"
    maxPrompts: 2
  transitions:
    next: "printSelectedAirport"
    actions:
      cancel: "handleMaxPromptsExceed"
```

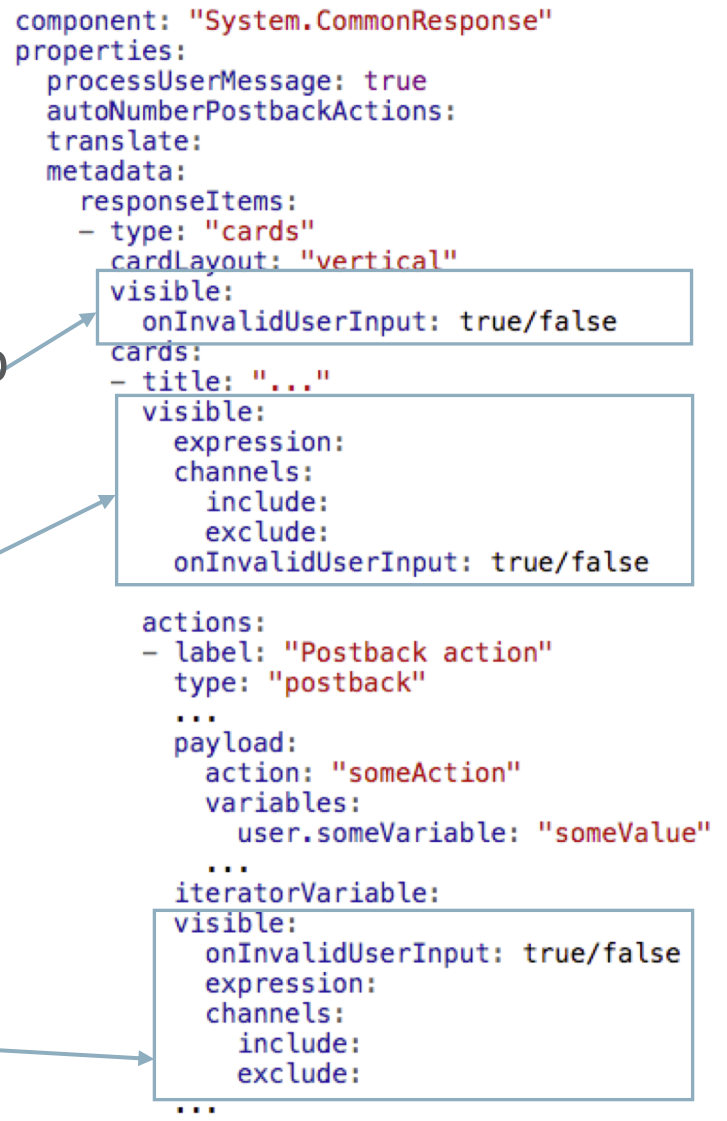
- Provide information about the problem when trapped in a validation loop
- *system.invalidUserInput?boolean*
  - Is set to true when a previous input failed validation
- Use Apache FreeMarker expression to conditionally show additional text

Topic agenda

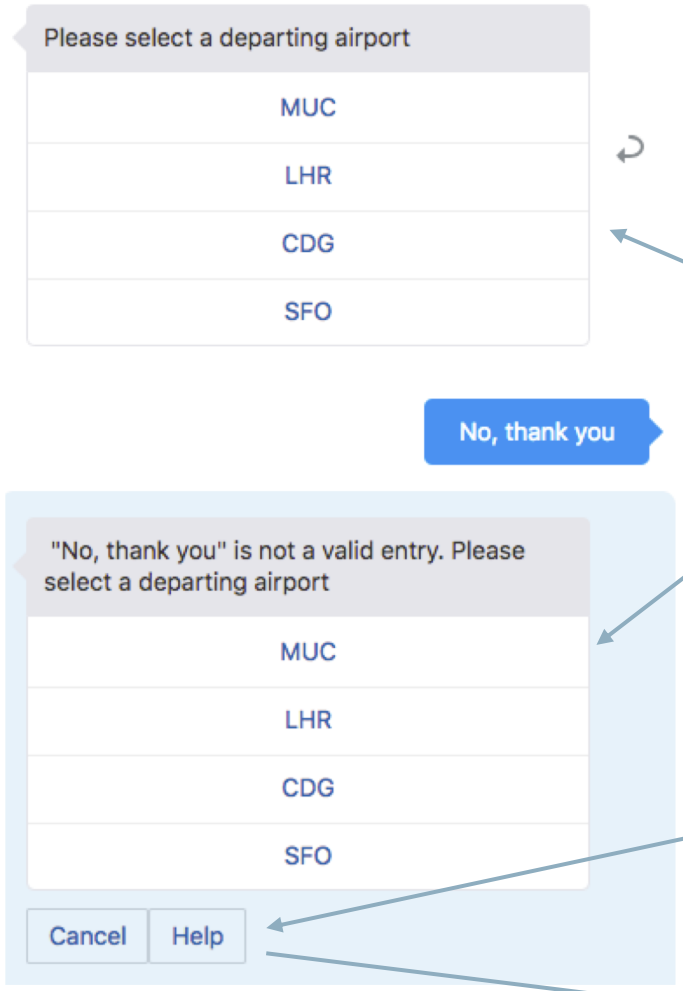
1. Escaping the validation loop
2. Assist users with prompts
3. Show visual aids
4. Handle free text input
5. Use data input forms
6. Handle out-of-order messages

About Common Response component

- Allows you to build richer bot responses
  - Supports variable, iResult properties for entity slotting
  - Supports maxPrompts property to escape the validation loop
  - Show/hide components (e.g. conditional help & cancel)
- onInvalidUserInput property
  - Automatically detects failed user entries
  - If set to true, displays content after failed use input
  - Available in the 'visible' node
    - On individual response items
    - On 'actions' and 'cards' nodes
    - On global actions buttons



Show cancel & help buttons after failed data input




```
selectAirport:
  component: "System.CommonResponse"
  properties:
    processUserMessage: true
    keepTurn: false
    variable: "airports"
    nlpResultVariable: "iResult"
    maxPrompts: 2
    metadata:
      responseItems:
      - type: "text"
        text: "<#if system.invalidUserInput?boolean> \"${system.message.messagePayload.text}\"
               is not a valid entry. </#if>Please select a departing airport."

      actions:
      - label: "${enumValue}"
        type: "postback"
        payload:
          variables:
            airports: "${enumValue}"
        iteratorVariable: "airports.type.enumValues"
      globalActions:
      - label: "Cancel"
        type: "postback"
        visible:
          # expression: "${system.invalidUserInput?boolean}"
          onInvalidUserInput: true
        payload:
          action: "exitComponentLoop"
      - label: "Help"
        type: "postback"
        visible:
          #expression: "${system.invalidUserInput?boolean}"
          onInvalidUserInput: true
        payload:
          action: "help"
  transitions:
    next: "printSelectedAirport"
    actions:
      exitComponentLoop: "exitComponentLoop"
      cancel: "handleMaxPromptsExceed"
      help: "selectAirportHelpState"
```

The Common Response component needs **at least one visible response item**.

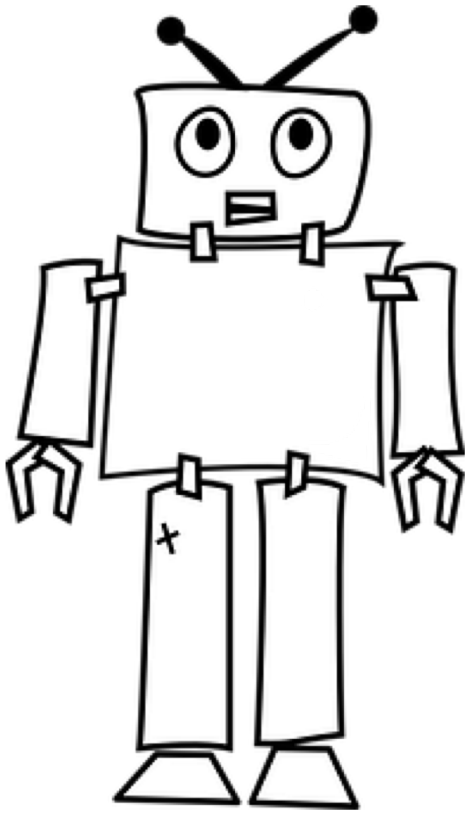


Image courtesy of pixabay.com

Topic agenda

1. Escaping the validation loop
2. Assist users with prompts
3. Show visual aids
4. Handle free text input
5. Use data input forms
6. Handle out-of-order messages

Free text input

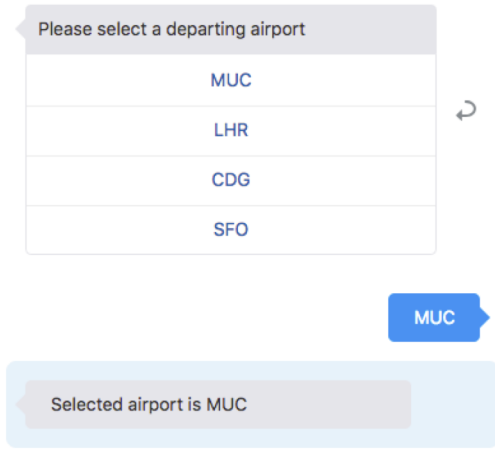
- Allow users to enter free text when selection displayed
- Options to avoid the validation loop
  - Set maxPrompts to 1 and use cancel action transition
  - Don't reference an entity type variable
    - Use context variable of type string
    - Don't set variable property but use postback actions (Common Response component)
- List, CommonResponse and other components provide textReceived message action transition
  - Triggered when free text is entered that does not match a valid component value

```
context:
  variables:
    airports: "AIRPORTS"
    iResult: "nlpresult"
    selectedAirports: "string"

states:
  selectAirport:
    component:
    properties:
      prompt: "Please select a departing airport"
      options: "${airports.type.enumValues}"
      variable: "selectedAirports"
    transitions:
      next: "printSelectedAirport"
    actions:
      textReceived: "handleTextReceived"
```

Using a string variable and textReceived action

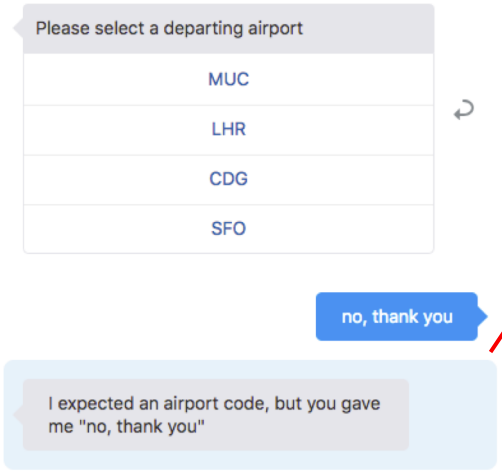
- String variable referenced from list's "variable" property
  - List validates user input against value displayed in the list
  - If user input matches list value, then value is saved in string variable and "next" transition is followed




```
context:
  variables:
    airports: "AIRPORTS"
    iResult: "nlpresult"
    selectedAirports: "string"

states:
  selectAirport:
    component:
    properties:
      prompt: "Please select a departing airport"
      options: "${airports.type.enumValues}"
      variable: "selectedAirports"
    transitions:
      next: "printSelectedAirport"
      actions:
        textReceived: "handleTextReceived"
```

- textReceived is called when user entered value does not match an entry in the list
  - User input is saved in string variable and also accessible from Apache FreeMarker expression



If a user provides **valid data input**, then the **textReceived action transition is not followed**. Instead the next transition (if set) or empty transition is followed.

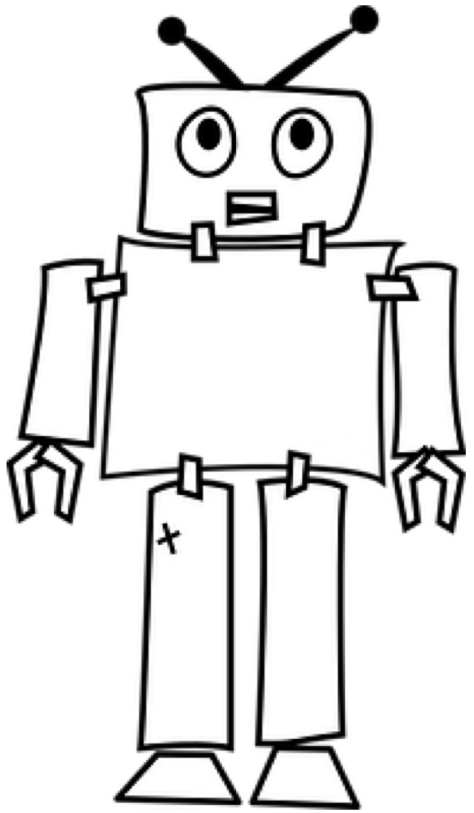
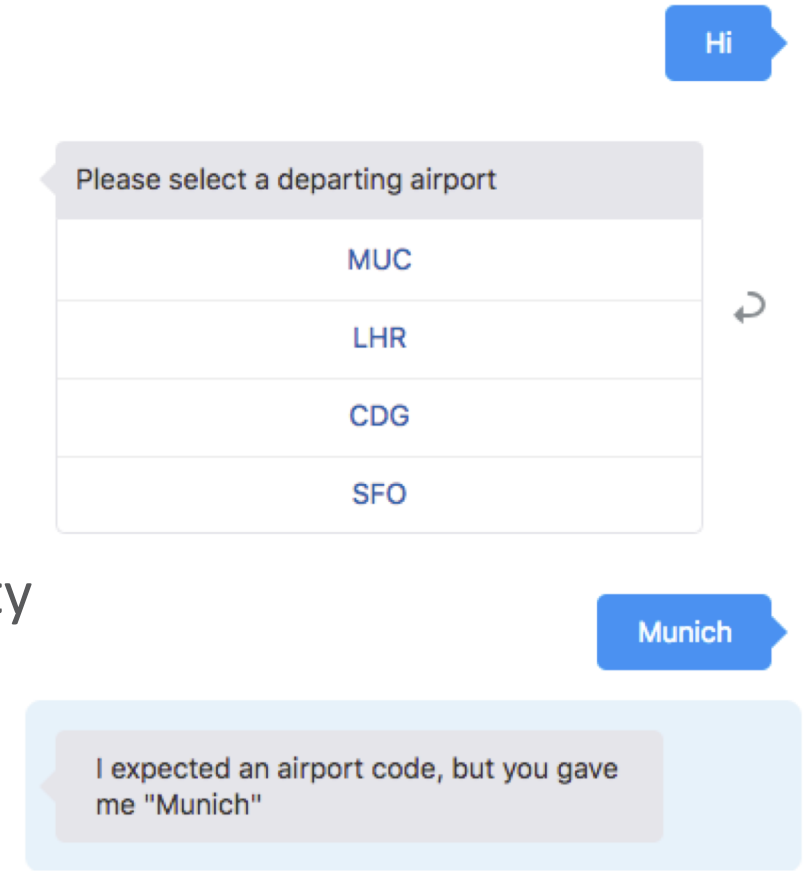


Image courtesy of pixabay.com

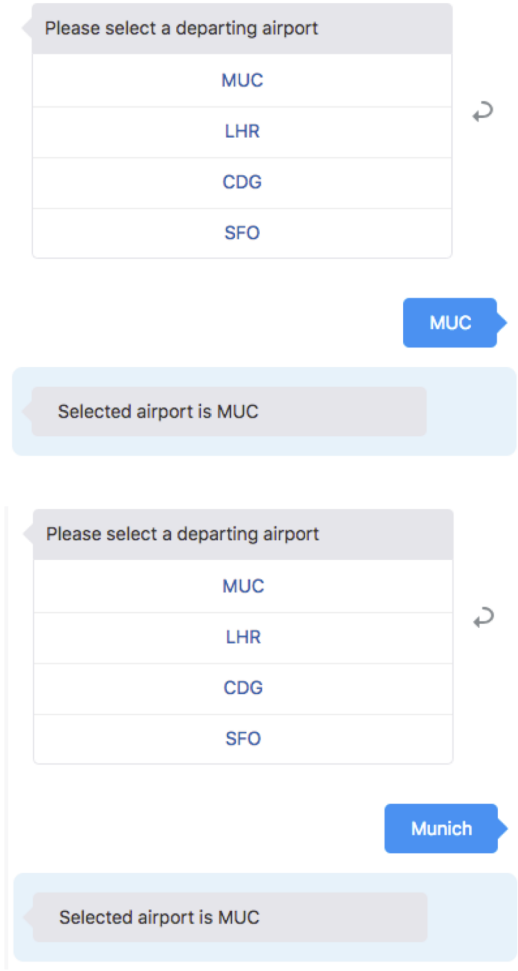
The downside of using string variables

- No entity slotting
  - No entity type referenced from component "variable" property
- "airports" entity variable not getting updated
- Only list validation performed
  - Validates user input against list option
  - E.g. "Munich" is a synonym defined on the value list entity
  - False positive. Though Munich is a valid entry for the entity, it is not leading to "next" transition
- Solution: Use an extra dialog flow step handle this



Use System.MatchEntity to validate user input

- List uses string variable to save user value
- Always navigates to state with System.MatchEntity component
- System.MatchEntity matches string variable value with entity by updating the entity variable
- If entity value is found, the continue as normal
- If no entity value is found, handle the free text input
  - e.g. route request to intent state



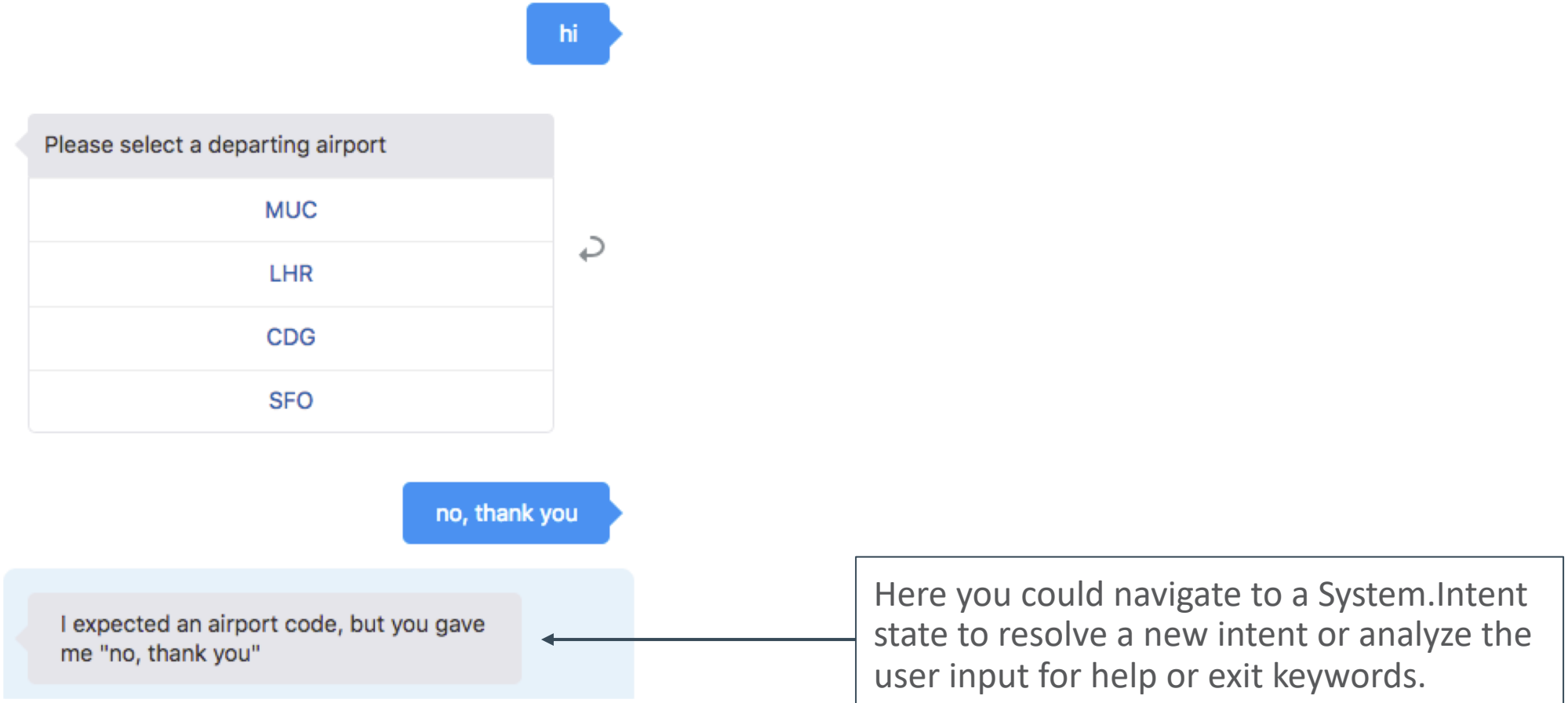

```
context:
  variables:
    airports: "AIRPORTS"
    iResult: "nlpresult"
    selectedAirports: "string"

states:
  selectAirport:
    component: "System.List"
    properties:
      prompt: "Please select a departing airport"
      options: "${airports.type.enumValues}"
      variable: "selectedAirports"
    transitions:
      next: "validateUserEntry"
      actions:
        textReceived: "validateUserEntry"

  validateUserEntry:
    component: "System.MatchEntity"
    properties:
      sourceVariable: "selectedAirports"
      variable: "airports"
    transitions:
      actions:
        match: "printSelectedAirport"
        nomatch: "handleTextReceived"

  printSelectedAirport:
    component: "System.Output"
    properties:
      text: "Selected airport is ${airports.value}"
    transitions:
      return: "done"
```

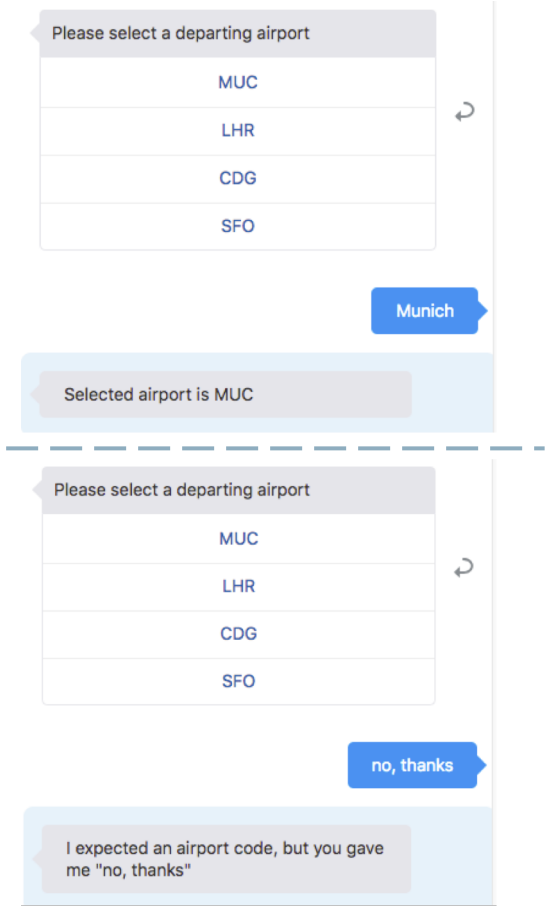
Free text entry still works



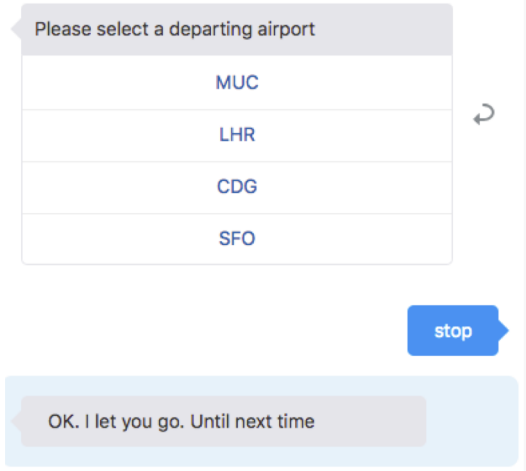
Here you could navigate to a System.Intent state to resolve a new intent or analyze the user input for help or exit keywords.

Handle help and exit keywords

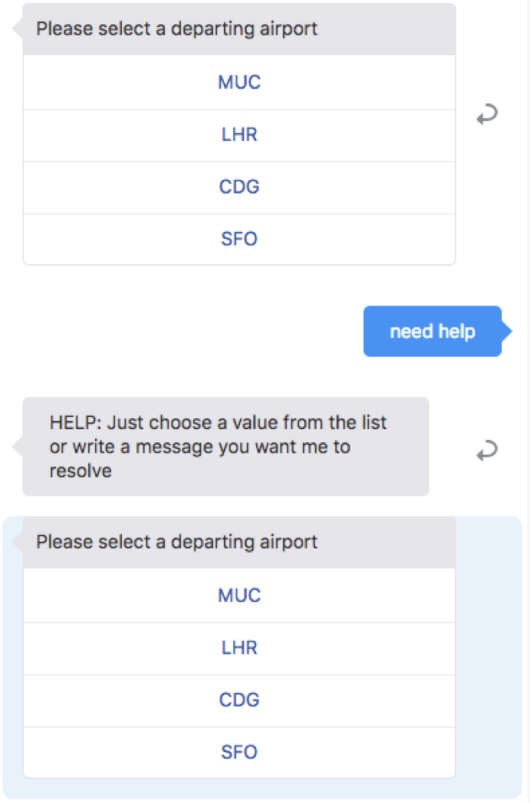
- Default Functionality



- Exit Keyword



- Help Keyword



Sample code using Apache FreeMarker expressions

```
selectAirport:
  component: "System.List"
  properties:
    prompt: "Please select a departing airport"
    options: "${airports.type.enumValues}"
    variable: "selectedAirports"
  transitions:
    next: "validateUserEntry"
    actions:
      textReceived: "checkForKeyWords"


checkForKeyWords:
  component: "System.Switch"
  properties:
    variable: "selectedAirports"
    source: ""
    values:
    - "exit"
    - "help"
  transitions:
    actions:
      help: "goHelp"
      exit: "goExit"
      NONE: "validateUserEntry"
```

"<#if ['q', 'quit','stop','bye']?seq_contains(selectedAirports.value)>**exit**
<#elseif ['help','huh','need help','h']?seq_contains(selectedAirports.value)>**help**
<#else>**validateUserEntry**</#if>"

Be aware of the Oracle Digital Assistant "Exit" system intent. **Make sure you use stop words that the system intent does not respond to.** If needed re-train "Exit" the system intent.

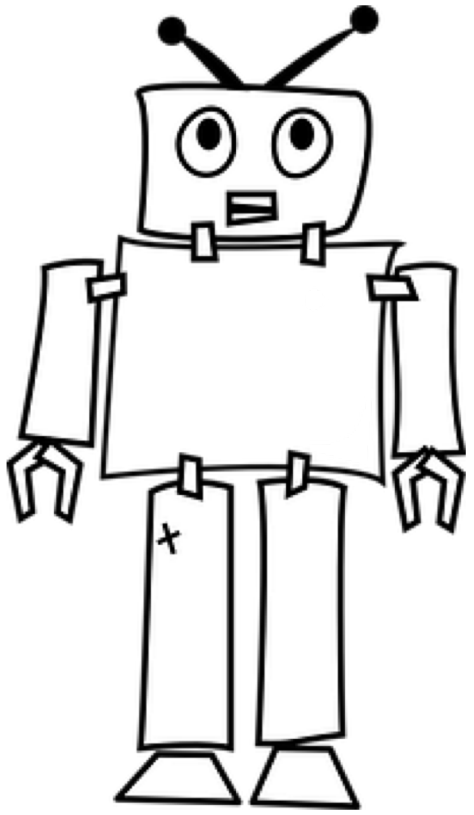


Image courtesy of pixabay.com

Topic agenda

1. Escaping the validation loop
2. Assist users with prompts
3. Show visual aids
4. Handle free text input
5. Use data input forms
6. Handle out-of-order messages

Imagine that **you are stuck in the middle of a very long conversation** and do not know what else to do. Would you like the bot to rewind the entire conversation? Certainly not.

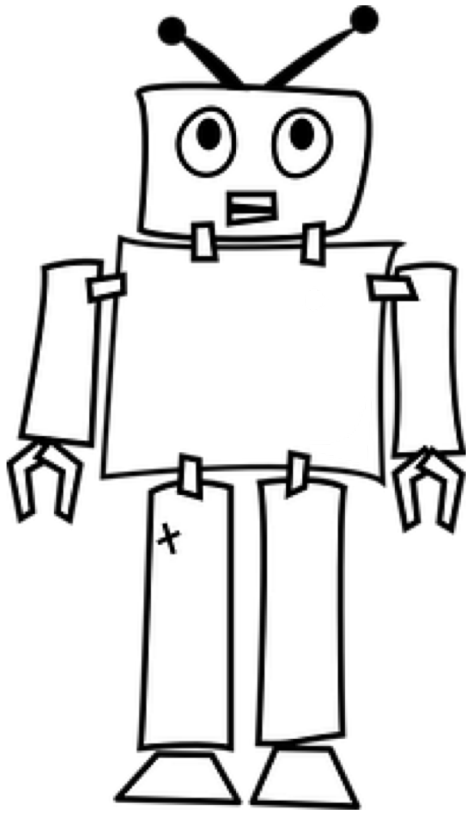


Image courtesy of pixabay.com

You Got Stuck.

Which cities and dates would you like to travel?

You can say things like "Fly from Riyadh to London from Jan 29 to Feb 3"

fly from amman

To which city/airport you will be flying to?

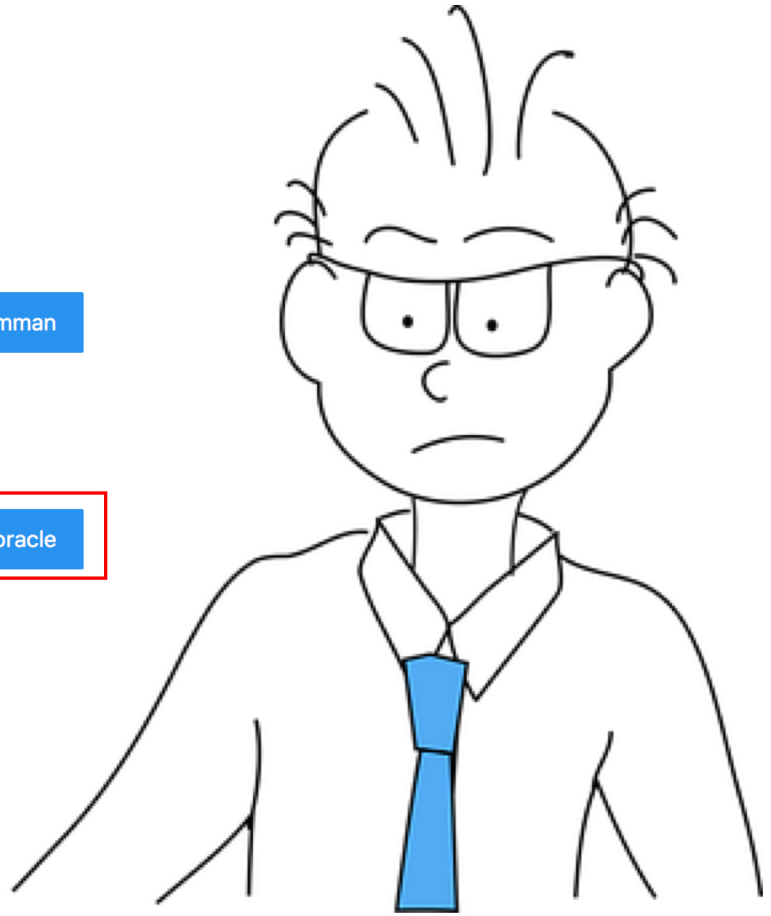
oracle

I apologize, but this does not appear to be a destination we fly to

To which city/airport you will be flying to?

Need Help?

Cancel



The Bot Knows How to Help You Out

Which cities and dates would you like to travel?

You can say things like "Fly from Riyadh to London from Jan 29 to Feb 3"

fly from amman

To which city/airport you will be flying to?

oracle

I apologize, but this does not appear to be a destination we fly to

To which city/airport you will be flying to?

Need Help?

Cancel

السعودية SAUDIA

From where you are traveling?

From: Amman

To:

When are you going?

Departure: 07/08/2018

Return: 07/09/2018

How many are going?

Adults: 1

Children: 0

Infants: 0

Please choose your ticket type

Economy

Go

Cancel

Display Webforms for structured data input

- Oracle Digital Assistant provides the System.Webview component for you to display web forms to users
  - Allows to quickly gather large sets of information from the user without going into a long conversation
  - Allows users to get unstuck by displaying a form that shows all the provided information along with the missing information
- Allow you to use rich web input elements
  - Sometime a user needs an environment he feels more comfortable with

Topic agenda

1. Escaping the validation loop
2. Assist users with prompts
3. Show visual aids
4. Handle free text input
5. Use data input forms
6. Handle out-of-order messages

Out-of-order Messages

- When in a bot conversation, users may scroll back in the conversation history and click on an action list or button
  - Bot receives user message that does not match the current state of the conversation
- Dependent on usecase, you may allow or not, out-of-order messages
- Oracle Digital Assistant default behavior is to allow out-of-order messages
  - Sensible default setting. May however not what you want
  - Default behavior can be changed and customized
- Out-of-order message handling is an important aspect in your conversation design
  - Only you know what's right

Integrated Cloud

Applications & Platform Services


ORACLE®