
ORACLE®

Oracle Digital Assistant

The Complete Training

Custom Component Design

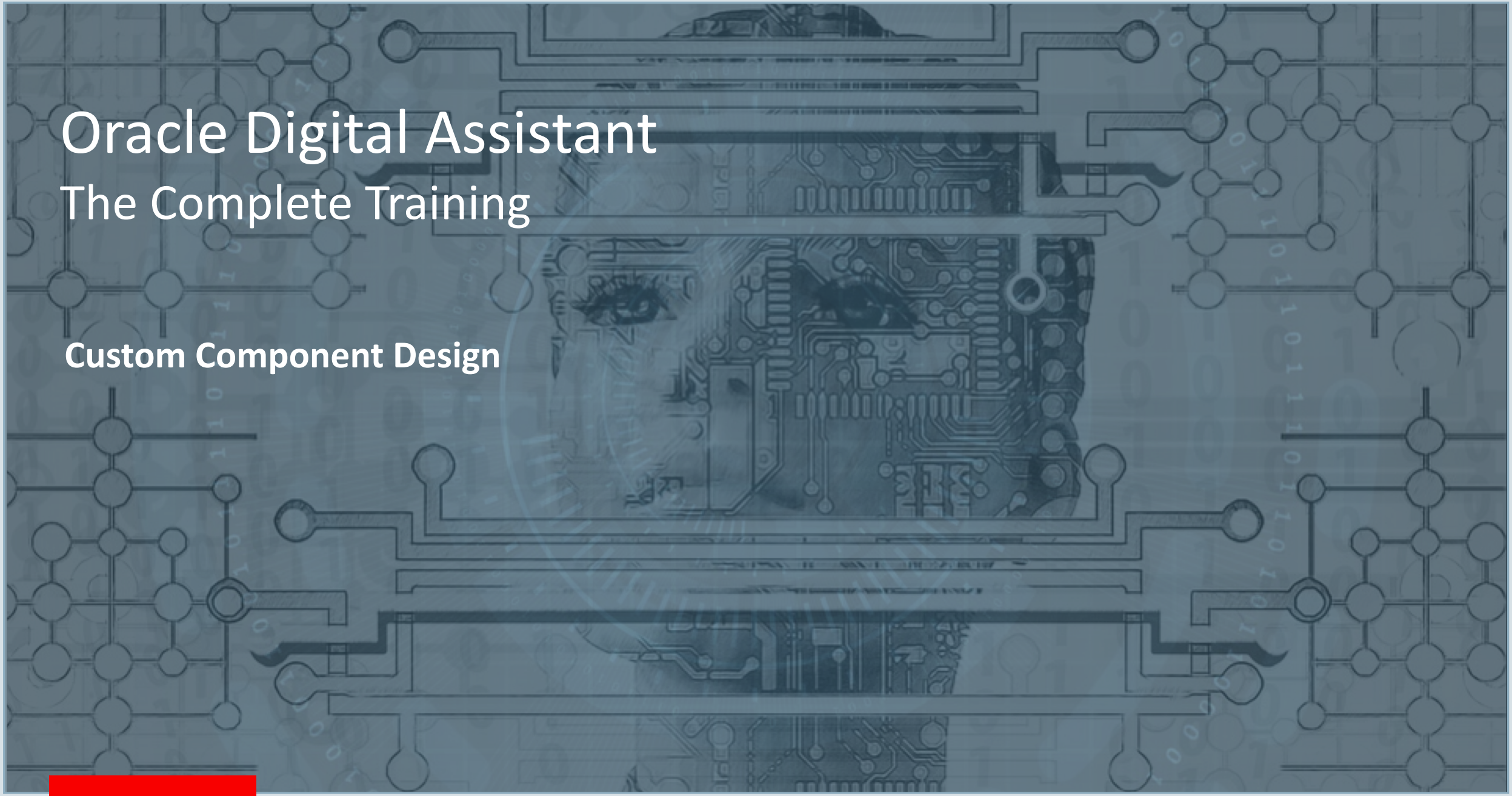


Image courtesy of pixabay.com

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.

Topic agenda

1. Message handling
2. Backend integration
3. Error handling
4. Writing responses vs. writing data
5. Custom component vs. CR component

Topic agenda

1. Message handling
2. Backend integration
3. Error handling
4. Writing responses vs. writing data
5. Custom component vs. CR component

"We built a custom component to display cards. **A click on an action does not trigger navigation** nor does it set the value of a variable"

Image src: pixabay

The word ***custom*** in custom component actually means '***on your own***'

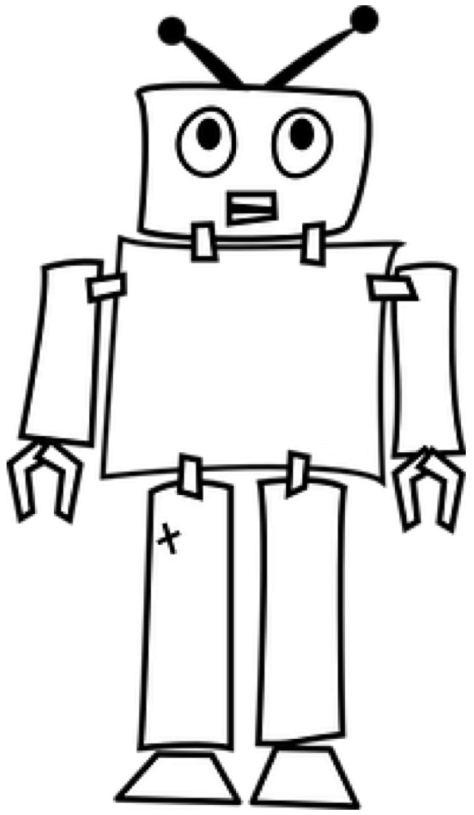


Image courtesy of pixabay.com

Creating components that interact with users

- Custom components that interact with users need to
  - Pause transition after the initial UI rendering
    - Not calling conversation.transition() ensures the custom component state remains current
    - A call to conversation.keepTurn(false) passes input control to the user
  - Manage state
    - Keep track of whether the component needs to render its UI or awaits user input
    - You can use simple or complex state token to keep track of UI and data states
- At the end of the component processing
  - Call conversation.transition(string) to continue dialog flow navigation
  - Using conversation.keepTurn(true) navigates without waiting for a user input

Option 1: manage state in a context variable

- Create and manage context variable in custom component
  - E.g. conversation.variable('_aUniqueStateTokenName_', stateValue);
- State token can be string or object (for complex state tracking)
- For every incoming message
  - Check state variable for what to do
  - Determine message type
- Context variable gets deleted when conversation ends with return statement

Option 2: add a token to an action payload

- Add extra attribute to payload when rendering buttons or action items
  - Payload defined a JSON object
    - { "orderId:"1234567", "_mystatetoken_":"value"}
- For every incoming message
  - Verify the bot message is an action payload
    - A call to conversation.postback()returns null if message is not an action
  - If an action message is found
    - Access payload (conversation.postback()) and check the state token

Example

Add token to payload

```
var _cancelLabel = "Cancel order";
var cancelAction = conversation.MessageModel().
                   postbackActionObject(_cancelLabel, null,{
    "orderAction": "cancel",
    "orderId": _orderId,
    "statetoken": _token
});
```

Check message and token

```
const _token = "advt.24hrs.flowers-token-gHy45123cD2z"

//check if it is a postback request and state token mat
if (conversation.postback()
    && conversation.postback().statetoken == _token) {
    //is postback issued by this component
    let _action = conversation.postback().orderAction;
    if (_action == 'ok') {
        conversation.transition("NoChangeRequest");
        conversation.keepTurn(true);
        done();
    } else if (_action == 'cancel') {
```

Checking whether the message type is "action" is not enough. You must **verify the token** because other components may also send action messages.

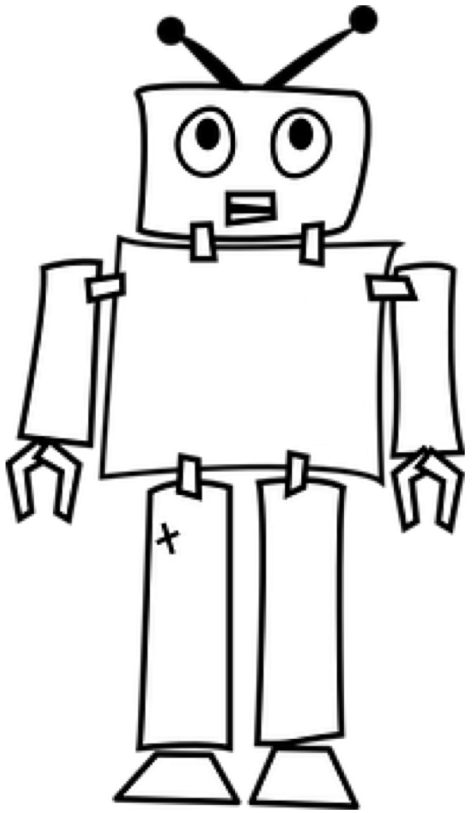


Image courtesy of pixabay.com

Context variable token vs. payload token

Context variable

- Exists for duration of conversation
  - Allows state to be tracked beyond multiple custom component uses
- Suitable for complex state tracking and memorizing of values
- Variable cannot be deleted from custom component

Payload

- One-time token
  - Not persisted in dialog flow
  - No 'housekeeping' required
- JSON string
  - Allows different token to be created for individual action items
- Lightweight

How to determine the type of a bot message

- `conversation.text()` returns text string or null
  - If(conversation.text()) { … handle text response ..}
- `conversation.attachment()` returns attachment string or null
  - If(conversation.attachment()) { … handle attachment response ..}
- `conversation.location()` returns location JSON object or null
  - If(conversation.location()) { … handle location response ..}
- `conversation.postback()` returns postback JSON object or null
  - If(conversation.postback()) { … handle potsback response ..}

A user who selects an item from a list of values or enters a list item name **sends an action message**. A user typing something that is not in the list sends **a text message**.

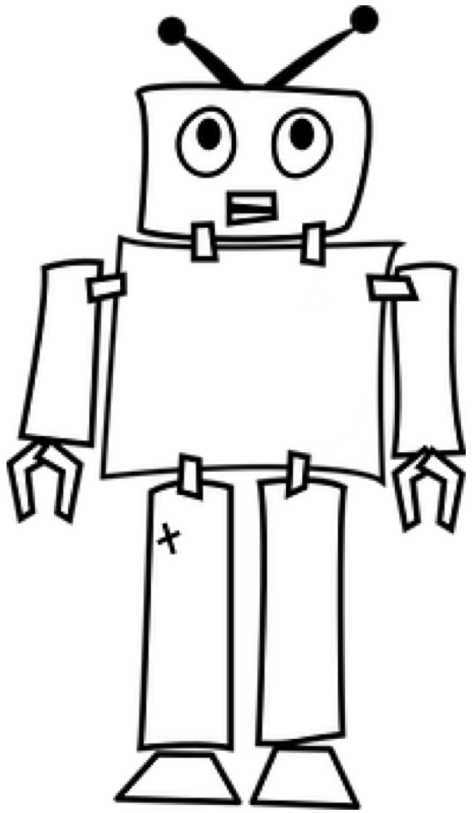


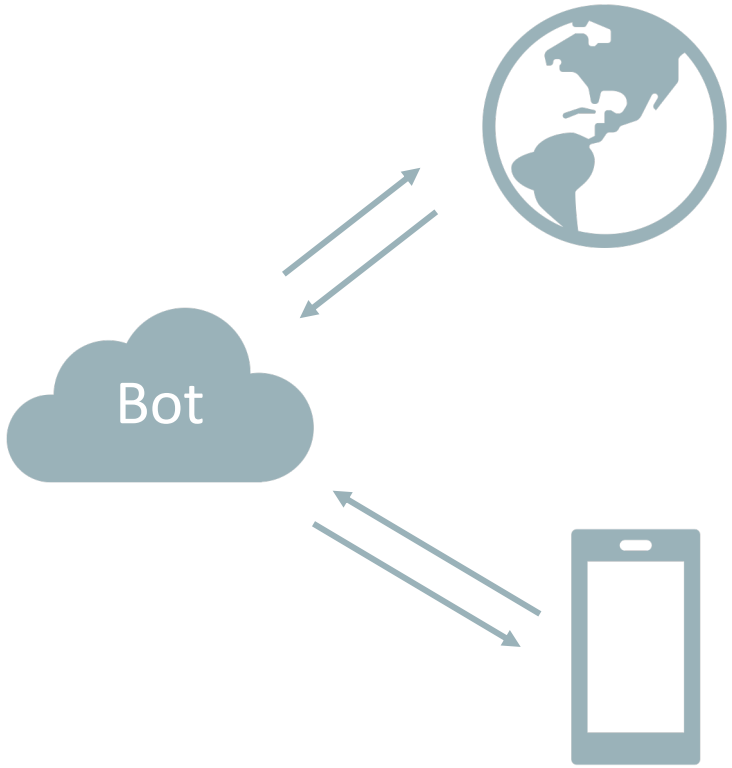
Image courtesy of pixabay.com

Topic agenda

1. Message handling
2. **Backend integration**
3. Error handling
4. Writing responses vs. writing data
5. Custom component vs. CR component

Backend integration depends on deployment

- Local custom component container
  - Backend access coded in custom component
  - Use Node modules like 'request' and 'https'
- Oracle Mobile Hub
  - Service access through connectors
  - Access to connector through Mobile Hub SDK
- Oracle application container cloud service
  - Same as for any other Node server
  - Backend access coded in custom component
  - Supports environment variables



Backend integration with local component container

Accessing OpenWeatherMap using the Node request module

```
var request = require('request');
…
const host = 'api.openweathermap.org';
const path = '/data/2.5/weather?q='+cityName+','+countryCode+'&appid='+appKey;
request('https://' + host + path, {json: true }, (err, res, body) => {
   var weatherResponse = body; //access JSON payload with weather info
   If (res.statusCode == 200) {
       …
       conversation.transition('success);
       done();
    } else {
       conversation.transition(error');
       done();
    }
});
```

Before using the "request" module, **investigate other options** too and select the Node module that suites your needs the best

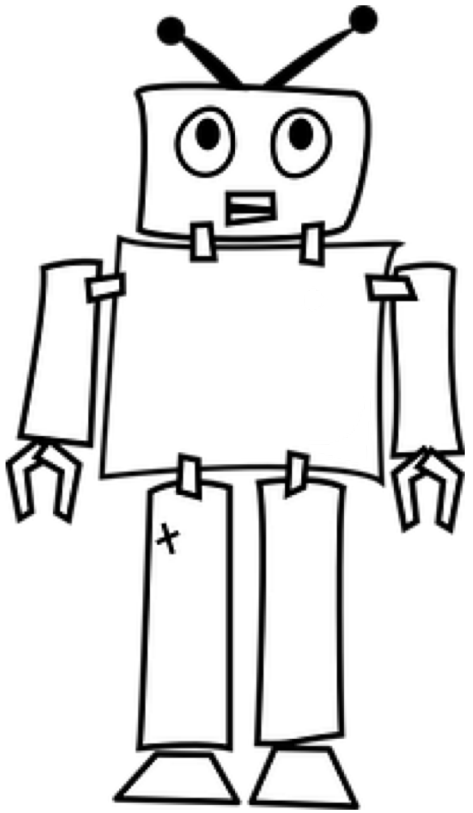


Image courtesy of pixabay.com

Topic agenda

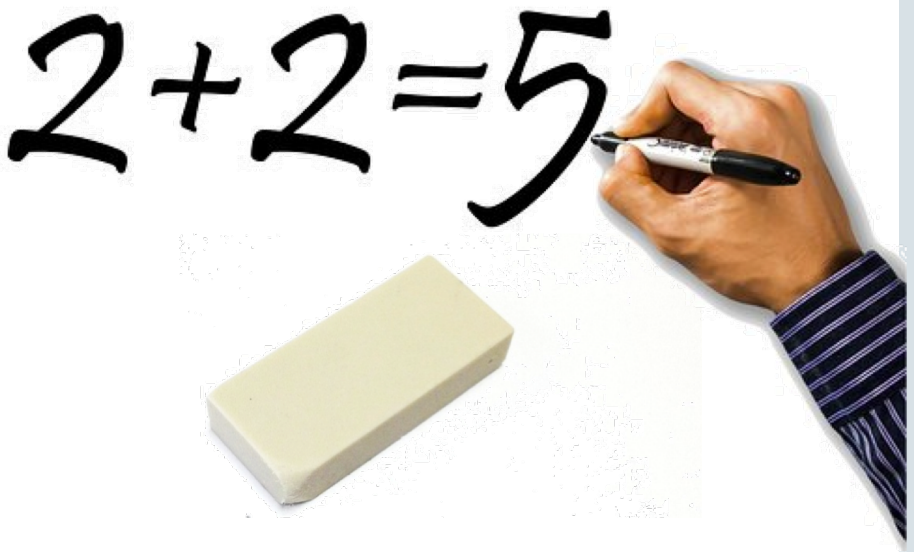
1. Message handling
2. Backend integration
3. Error handling
4. Writing responses vs. writing data
5. Custom component vs. CR component

Possible custom component failures

- Operational
  - Backend access
    - Authentication & Authorization failure, Service unavailable
  - Application errors
    - Everything that prevents successful processing
    - JSON parsing errors, missing input parameter values
- Programming
  - Missing dependent files
  - Syntax problems

Error handling strategy

- Use action transitions for operational issues
  - Return action string that indicate the problem
  - Bot designers handle actions to states in dialog flow
- Use error transition for errors that cannot be handled by bot designer
  - Create a JS error object
  - Pass error object as argument to the done() callback call

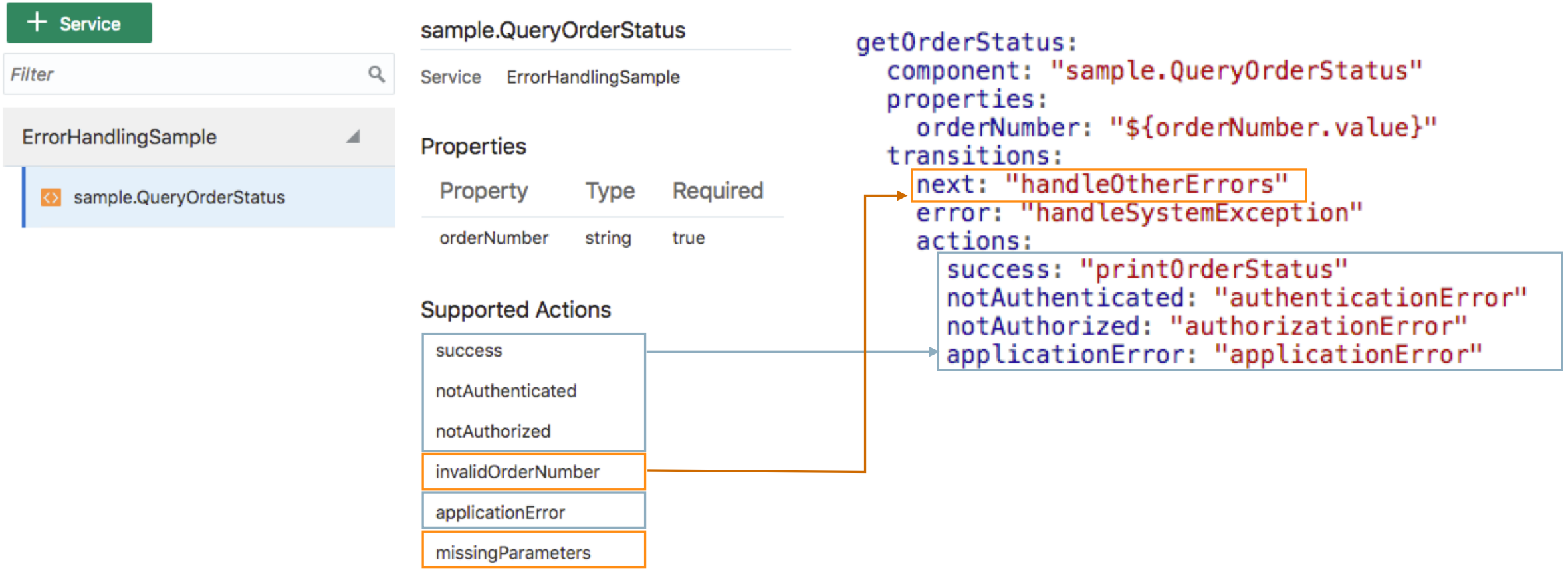


Error handling responsibilities by persona

- Custom component designer
  - Defines transition action for errors that the bot designer can handle
  - Defines transition action for component success
  - Throws error for all exceptions a bot designer cannot handle
- Bot designer
  - Defines error transition on component state to handle exceptions
  - Defines action transitions for all errors he wants to handle
  - Defines a "next" transition to handle application errors the bot designer doesn't want to deal with

Example for error handling using action strings

Design time view in skill



Example for error handling using action strings

Custom component code

```
invoke: (conversation, done) => {
    //...
    if(errorType == 'authentication'){
      conversation.transition("notAuthenticated");
      done();
    }
    else if(errorType == auhorization'){
      conversation.transition("notAuthorized");
      done();
    }
     else  {
      conversation.transition("applicationError");
      done();
    }
  …
  }
```

```
getOrderStatus:
  component: "sample.QueryOrderStatus"
  properties:
    orderNumber: "${orderNumber.value}"
  transitions:
    next: "handleOtherErrors"
    error: "handleSystemException"
    actions:
      success: "printOrderStatus"
    notAuthenticated: "authenticationError"
    notAuthorized: "authorizationError"
    applicationError: "applicationError"
```

Using the error transition for errors that cannot be handled

- It may be that failures can be caught but not handled
  - Use error transition
- Error transition is invoked from custom component by passing an Error object as an argument to the done() callbacl
  - Error state name can be accessed in dialog flow using ${system.errorState} expression
  - If there no error transition handler defined then the system error handler is used

```
if(errorType == 'exception'){
  var err = new Error("Serious Exception ");
  err.name = 'badRequest';  //optional
  done(err);
}
```

Elevator manufacturers have three ways of dealing with a fault: making an emergency call, applying the brakes and keeping people safe in the cabin, or dropping the cabin so that it quickly returns to a solid ground

Think about it next time before eagerly making a call to the error transition

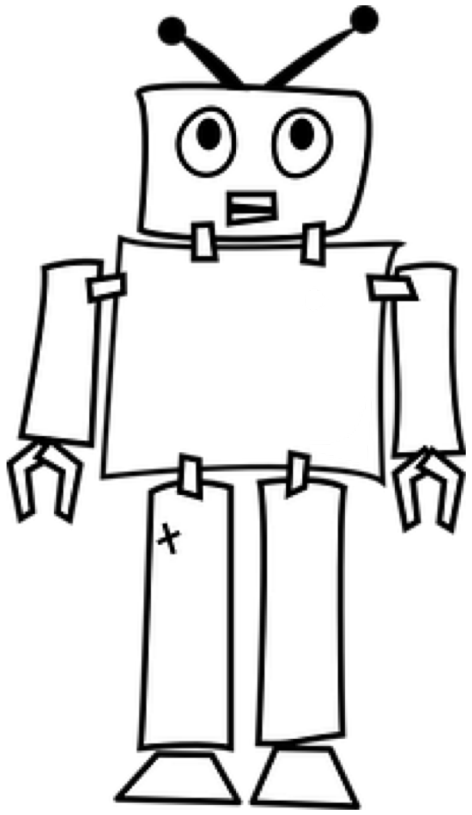


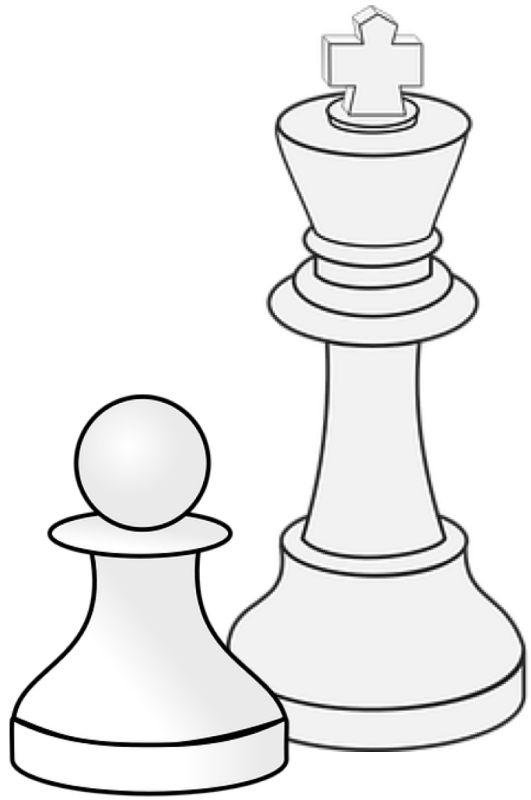
Image courtesy of pixabay.com

Topic agenda

1. Message handling
2. Backend integration
3. Error handling
4. Writing responses vs. writing data
5. Custom component vs. CR component

Custom component response strategies

- Two Broad Strategies
  - Render Response to Messenger Client
    - Use Conversation Message Model
    - Handle complete task
  - Save Data in Context Variable
    - Render data using CRC component



Writing data to a context variable

- PRO
  - Smaller response payload
  - Data saved in context variable until variable is reset or dialog flow is exit
  - Data can be auto-translated when displayed by system components
  - Data can be used with message bundles
- CON
  - No option available to describe the data structure to bot designers
    - Makes it hard to share the component
  - Requires bot designer to know how to use the component
    - E.g. need to create context variable

Writing response to message channel

- PRO
  - No need to document response data structure
  - Custom component developers have full control over the rendered UI
  - Component can be designed as "black box" for easy reuse
- CON
  - Component response is more verbose compared to when just sending data
  - Data objects cannot be cached
    - Information needs to be re-queried if needed more often
  - Multi-language translation must be handled by the component

Be Mindful when saving data in context variables**. A bot is not a database**

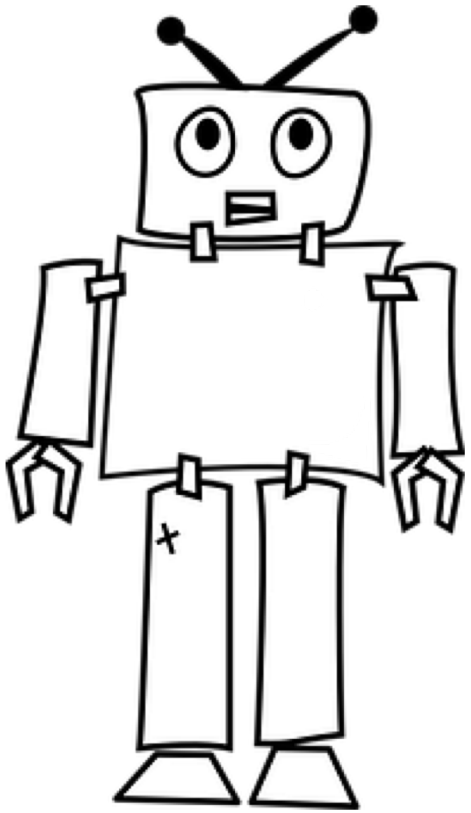


Image courtesy of Farbentek at FreeDigitalPhotos.net

Topic agenda

1. Message handling
2. Backend integration
3. Error handling
4. Writing responses vs. writing data
5. Custom component vs. CR component

CR component features missing for custom components

Features you would have to code for

- Composite bot responses
- Composite bag entity support
- Entity validation
- Page ranging
- Entity slotting
  - variable
  - nlpResultVariable
- Auto translation

When to use the CR component

- Whenever possible
- Reduces risk of code changes due to SDK changes
- Use with data saved in context variables
  - Data in context variables is cached for a conversation
  - Can be used more than once
  - Has a smaller payload when passed to the bot

When to render the UI from a custom component

- When the custom component is a "closed" system
  - Handling of a complete a task
- When a component needs to go into longer "user interactions"
  - E.g. in a questionair
- When a component should be reused across bots
  - Avoiding dialog flow dependencies

Integrated Cloud

Applications & Platform Services


ORACLE®

Oracle Digital Assistant Hands-On

TBD