

ORACLE®

Oracle Digital Assistant

The Complete Training

Webview component

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

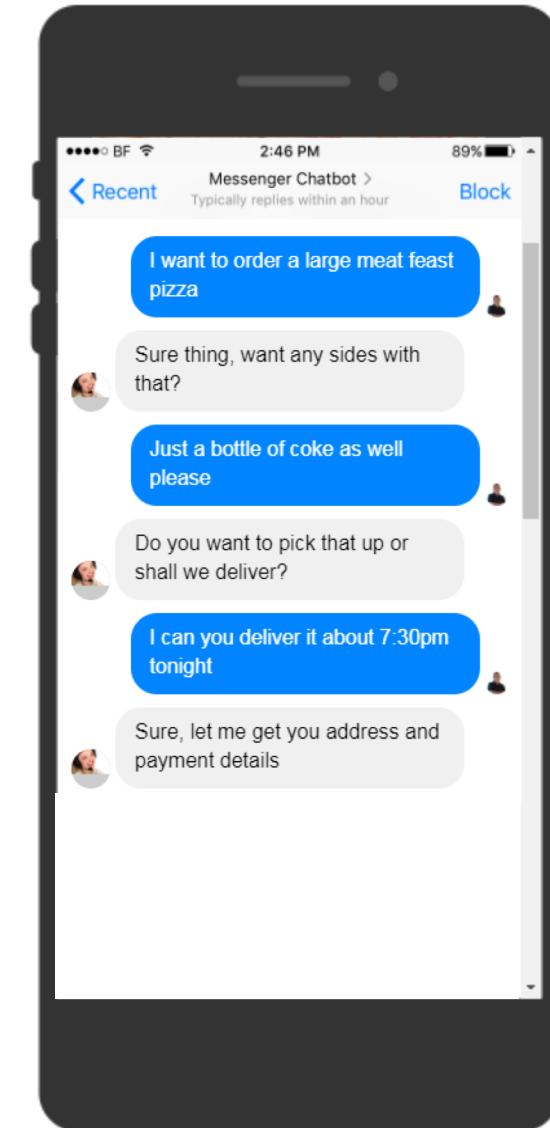
- 1 ➤ When conversation is not enough
- 2 ➤ System.Webview
- 3 ➤ Local web applications
- 4 ➤ Remote web applications
- 5 ➤ Remote web app vs. local web app

Topic agenda

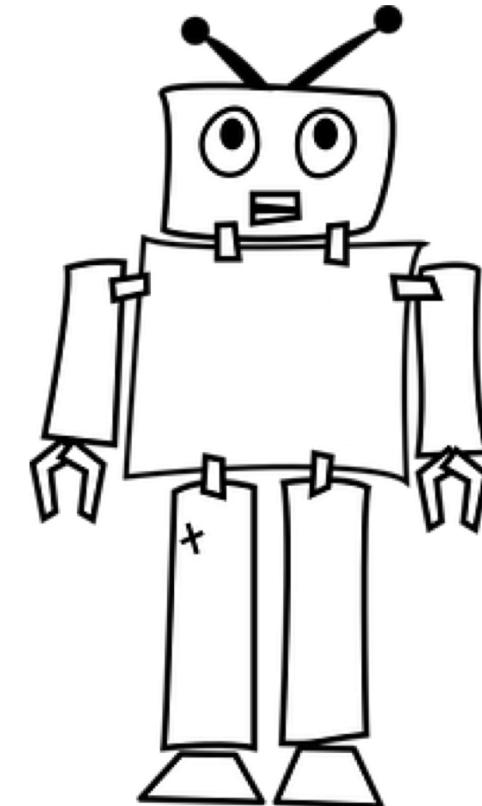
- 1 ➤ When conversation is not enough
- 2 ➤ System.Webview
- 3 ➤ Local web applications
- 4 ➤ Remote web applications
- 5 ➤ Remote web app vs. local web app

A typical conversation

- Natural language conversation
 - Unstructured
 - Free flowing
 - Conversational



Not all use cases are perfect matches
for the conversational channel.



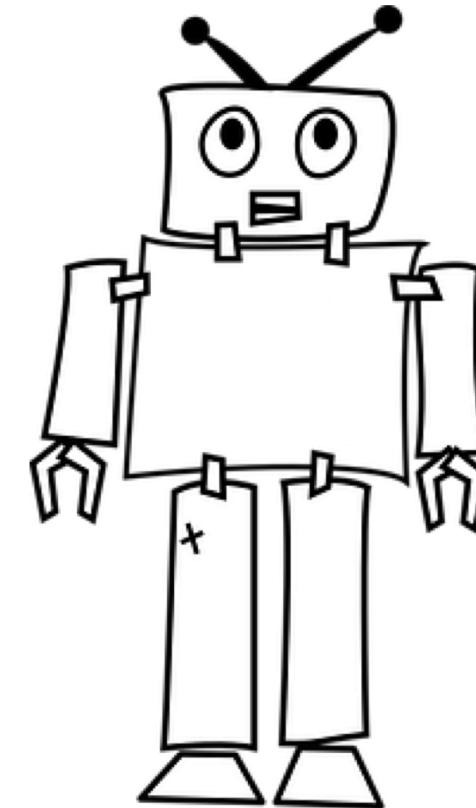
Limitations of the conversational channel

- Long conversations are difficult to have in a chat
 - Users have a limited attention rate and are easily distracted
 - Hard to change user input at a later point in a conversation
 - Lack of support for structured data input
- UI limitations
 - Missing widgets (e.g. date picker or calendar, LOV)
 - No type-ahead support
- Display of sensitive information
 - No way to obfuscate password or credit card details

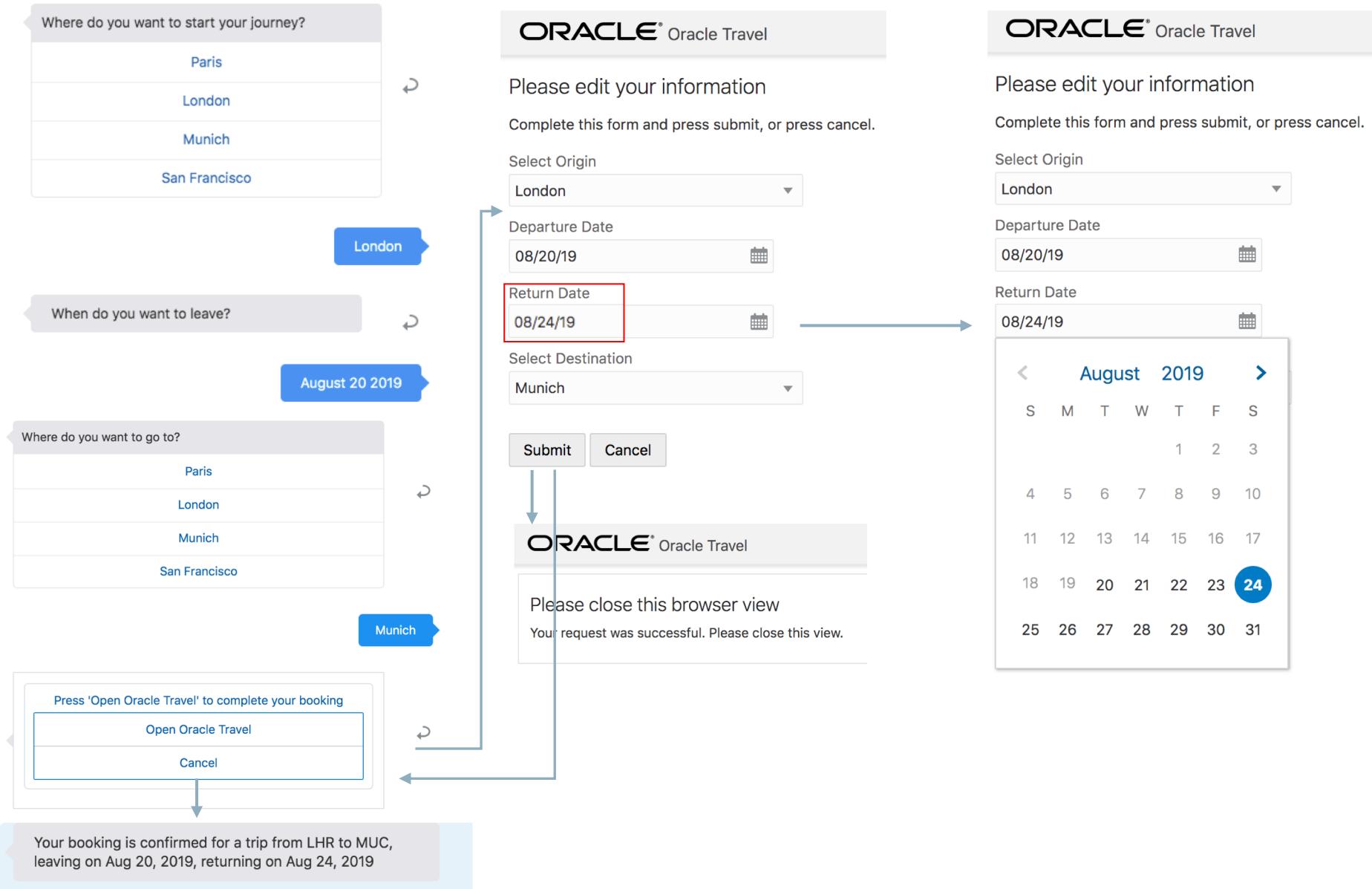


Humans use whiteboards and lists if conversation is not enough.

So what's the equivalent for chatbots?



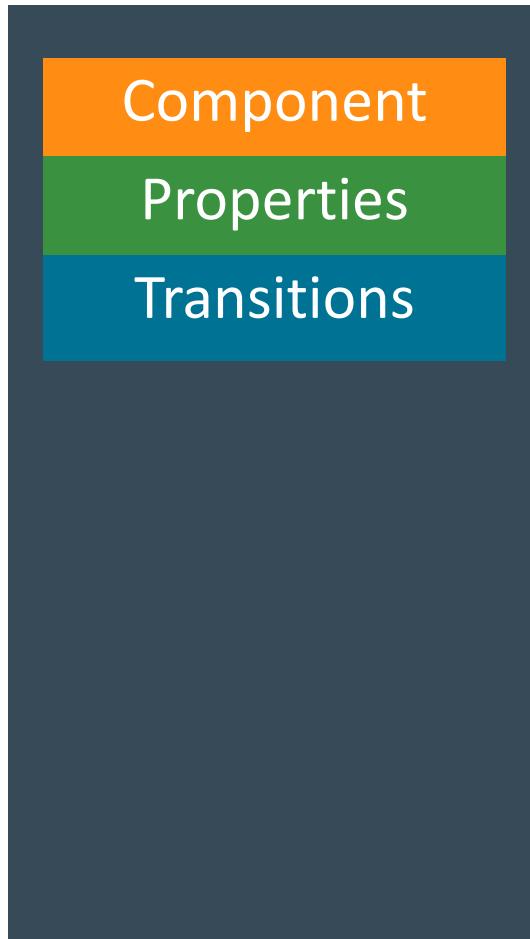
Example Oracle Travel



Topic agenda

- 1 ➤ When conversation is not enough
- 2 ➤ System.Webview
- 3 ➤ Local web applications
- 4 ➤ Remote web applications

System.Webview component overview



- Displays link to launch web application in separate browser tab (web) or webview (mobile)

The diagram illustrates the interaction between a list component and a webview component. On the left, a list component displays travel destinations: Paris, London, Munich, and San Francisco. A blue arrow points from the 'Munich' item to a modal dialog. This dialog contains the text 'Press 'Open Oracle Travel' to complete your booking' and two buttons: 'Open Oracle Travel' and 'Cancel'. A blue arrow points from the 'Open Oracle Travel' button to a webview component on the right. The webview displays the 'ORACLE® Oracle Travel' interface, which includes fields for 'Select Origin' (set to London), 'Departure Date' (set to 08/20/19), 'Return Date' (set to 08/20/19), and 'Select Destination' (set to Munich). It also features 'Submit' and 'Cancel' buttons at the bottom. A blue arrow points from the 'ORACLE® Oracle Travel' header back to the list component.

Where do you want to go to?

Paris
London
Munich
San Francisco

Munich

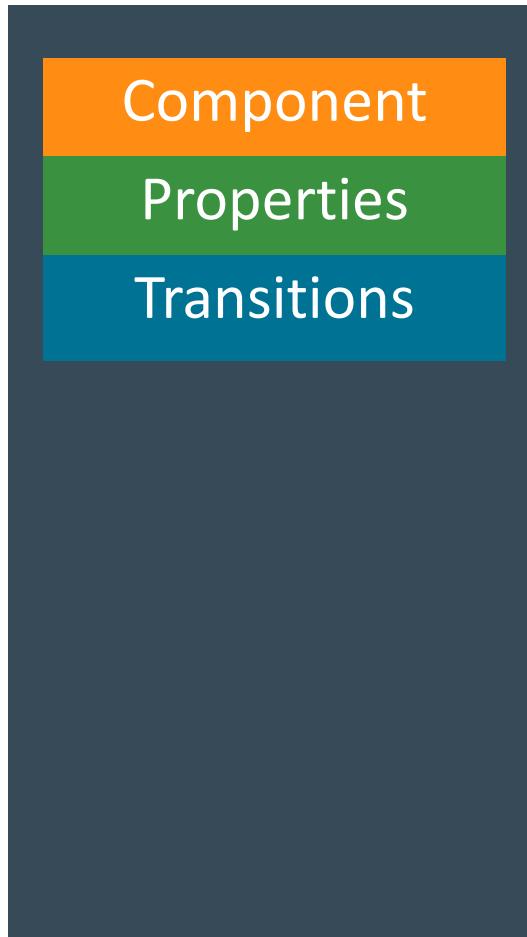
Press 'Open Oracle Travel' to complete your booking

Open Oracle Travel
Cancel

ORACLE® Oracle Travel

Please edit your information
Complete this form and press submit, or press cancel.
Select Origin
London
Departure Date
08/20/19
Return Date
08/20/19
Select Destination
Munich
Submit Cancel

System.Webview component overview



- Displays link to launch web application in separate browser tab (web) or webview (mobile)
 - Remote web application
 - Locally deployed Single Page Application (SPA)
- Passes parameters from bot to web application (optional)
- Web application uses callback URL to return to bot
 - Optionally, passes data back

Using the component template

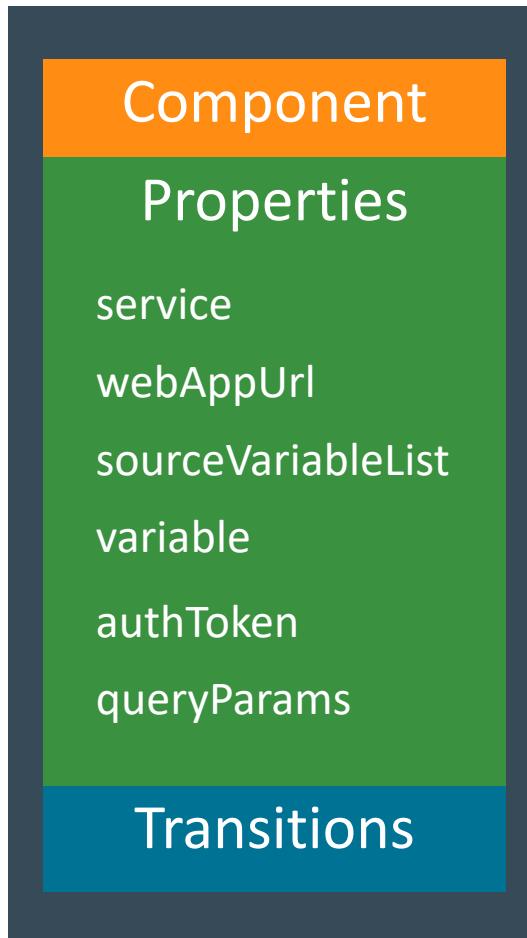
The screenshot shows the Oracle Bot Service interface. On the left, there's a sidebar with icons for Components, Variables, and Settings. A green button labeled '+ Components' is at the top of the main content area. Below it, a modal window titled 'Select a Component Type' lists four categories: Control, Language, Security, and User Interface. The 'User Interface' category is highlighted with a blue border. In the bottom right corner of this modal, there's a small 'X' icon.

On the right side, a detailed view of the 'User Interface' component template is shown. It has a title bar with 'User Interface' and 'Component Template' and a close 'X' icon. The template content is a code block:

```
webview:  
  component: "System.Webview"  
  properties:  
    # sourceVariableList is a comma-delimited list of user or  
    context variable names. These variable names are the parameters which  
    values will be made available to the webview. These variables are the  
    input parameters for the webview  
    sourceVariableList:  
      # variable refers to name of a single variable that will  
      contain a string value. This variable is the output result of the  
      webview that is used to identifies the callback payload. When the bot  
      user completes the interactions with the webview, the webview can  
      provide a user gesture to send a callback with a payload to the bot.  
      The payload will then be stored in this variable. The value in this  
      variable can be accessed at a later point in the dialog flow
```

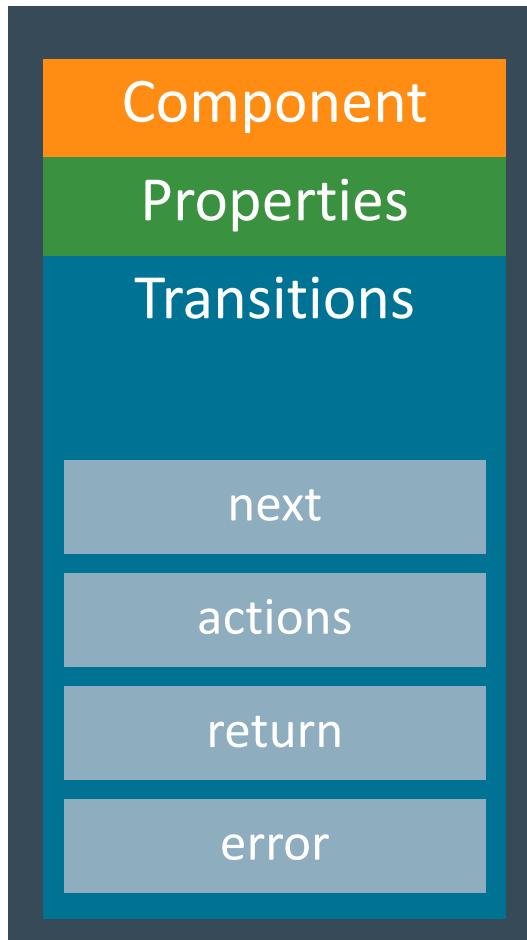
On the left side of this detailed view, there's a sidebar with links: Interactive, List - set action, List - set variable, Output, Resolve entities, Text, and Webview. The 'Webview' link is highlighted with a blue background. At the bottom of the detailed view, there are buttons for 'Insert After', 'Remove Comments' (with a toggle switch), and 'Apply'.

System.Webview component properties



- service
 - Named configuration for local or remote app access
- webAppUrl
 - Remote web application URL (legacy property)
- variable
 - receives data object returned from web application (callback)
- authToken
 - If remote application requires authentication
- queryParams (optional)
 - JSON formatted key-value pairs added to webAppUrl

System.Webview component transitions

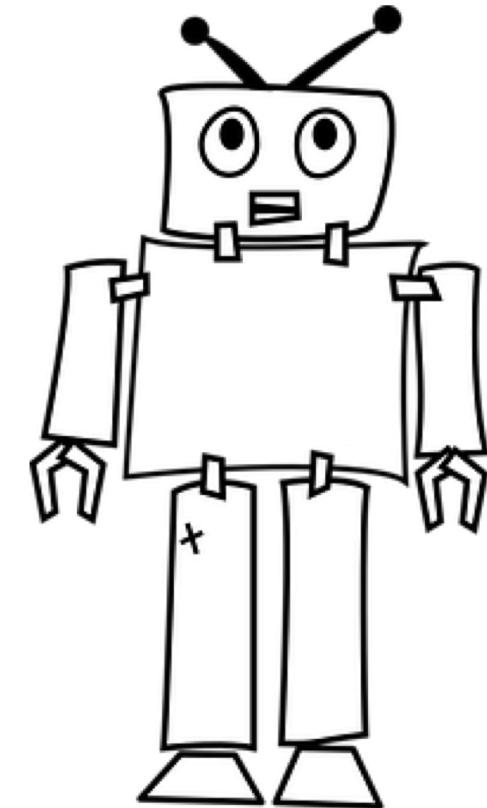


- Use **next** transition to navigate to next state upon web application success
- Use **return** transition to exit conversation upon web application success
- Use **error** transition to handle component exceptions
- Use **actions** transitions
 - **Cancel** to handle case where users presses "cancel" link
 - **textReceived** to handle case in which user types a text

Topic agenda

- 1 ➤ When conversation is not enough
- 2 ➤ System.Webview
- 3 ➤ Local web applications
- 4 ➤ Remote web applications
- 5 ➤ Remote web app vs. local web app

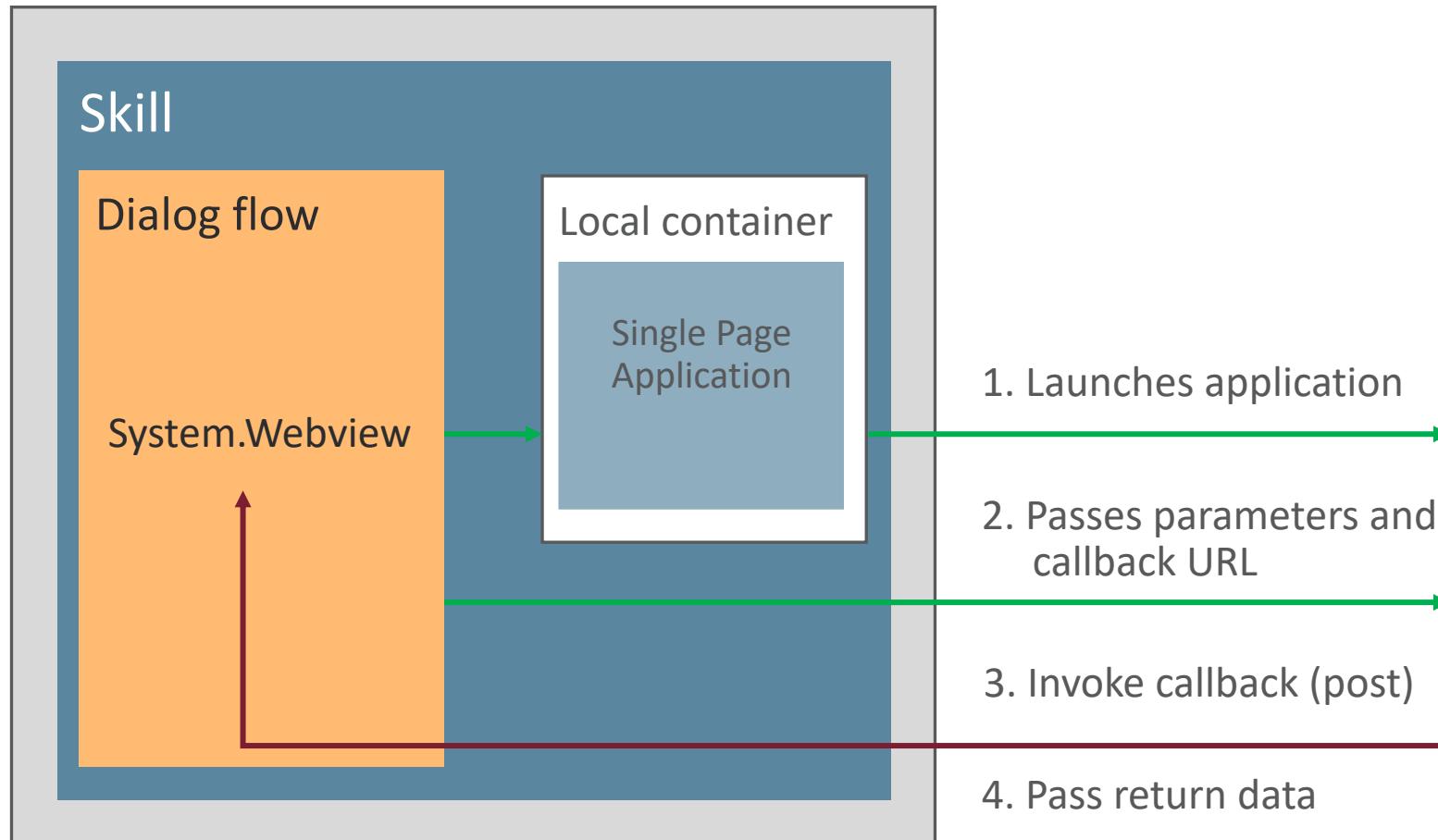
Like custom components, single page applications can be hosted within a skill



Single Page Application (SPA)

- Client side web applications that load a single HTML page
 - Dynamically updated in response to user interaction
 - No constant page reloads
 - Uses JavaScript, Ajax and HTML5
- Frameworks
 - Oracle JET, Oracle Visual Builder Cloud Service, Angular, React, etc.
 - Model–View–ViewModel (MVVM) architectures
 - Separation of data logic from UI definition and UI logic

Local invocation architecture

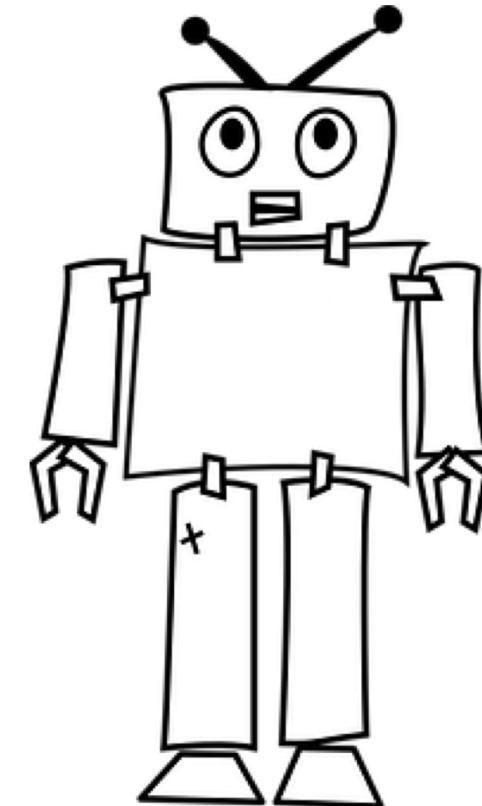


index.html

A screenshot of the "index.html" page. It features a "Browser / Webview URL" input field at the top. Below it are three "label" input fields. In the bottom right corner, there is a "Submit" button.

Parameters passed from the skill to
the web application are accessible
through JavaScript

```
window.webviewParameters['parameters'];
```

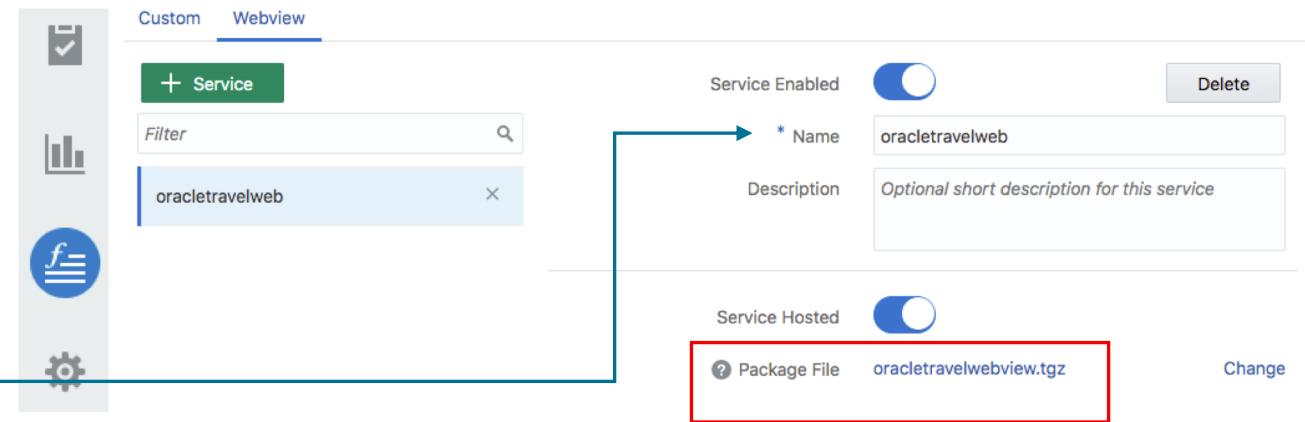


SPA Application Reference

Skill

Dialog Flow

```
callWebview:  
  component: "System.Webview"  
  properties:  
    service: "oracletravelweb"  
    sourceVariableList: "origin,destination"  
    variable: "webviewresponse"  
    prompt: "Press 'Open Oracle Travel' ..."  
    linkLabel: "Open Oracle Travel"  
    cancelLabel: "Cancel"  
  transitions:  
    next: "handleResponse"  
  actions:  
    textReceived: "onCancel"  
    cancel: "onCancel"
```



- Package web application
 - web folder
 - tar -zcvf <name>.tgz *
- Upload per drag and drop

SPA Application Parameter Passing

Skill

Dialog Flow

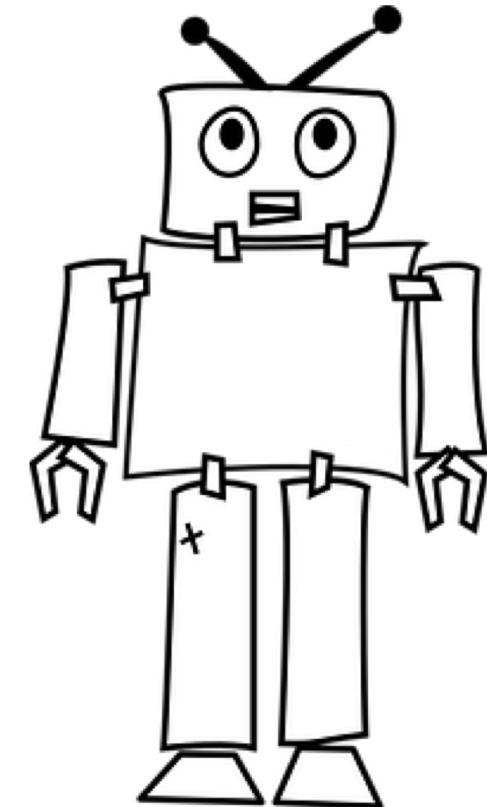
```
callWebview:  
  component: "System.Webview"  
  properties:  
    service: "oracletravelweb"  
    sourceVariableList: "origin,destination"  
    variable: "webviewresponse"  
    prompt: "Press 'Open Oracle Travel' ..."  
    linkLabel: "Open Oracle Travel"  
    cancelLabel: "Cancel"  
  transitions:  
    next: "handleResponse"  
  actions:  
    textReceived: "onCancel"  
    cancel: "onCancel"
```

```
{ parameters: [  
  {  
    "key": "origin",  
    "value": "LHR"  
  }, {  
    "key": "destination",  
    "value": "SFO"  
  }, {  
    "key": "webview.onDone",  
    "value": "http://<url>/connectors/v1/callback?state=cb5443...2c"  
  }  
]
```

Single Page Application

```
let webviewParameters = window.webviewParameters !=  
null ? window.webviewParameters['parameters'] : null;
```

The **webview.onDone** parameter is automatically added to the payload and passes the skill's **callback URL** property to the web application



SPA Application Response

Skill

Dialog Flow

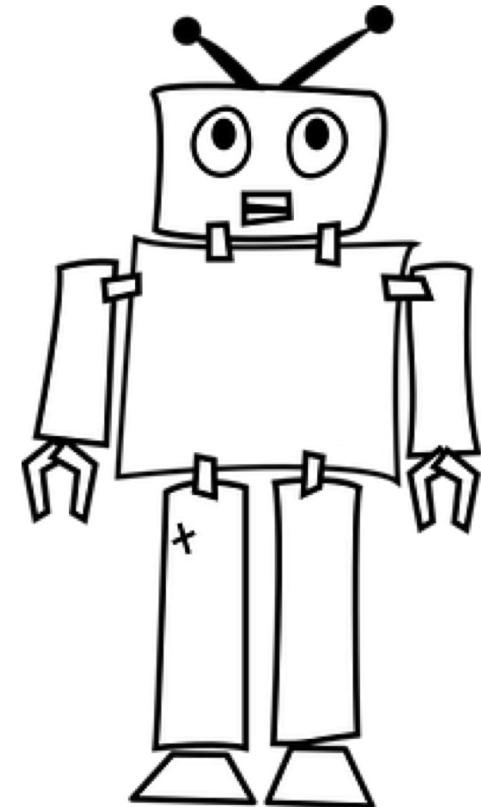
```
callWebview:  
  component: "System.Webview"  
  properties:  
    service: "oracletravelweb"  
    sourceVariableList: "origin,destination"  
    variable: "webviewresponse"  
    prompt: "Press 'Open Oracle Travel' ..."  
    linkLabel: "Open Oracle Travel"  
    cancelLabel: "Cancel"  
  transitions:  
    next: "handleResponse"  
  actions:  
    textReceived: "onCancel"  
    cancel: "onCancel"
```

```
let data = {};  
data.origin = self.origin();  
data.destination = self.destination();  
data.departureDate = (new Date(self.departureDate())).getTime();  
data.returnDate = (new Date(self.returnDate())).getTime();  
let webViewCallback = queryParameters.get('webview.onDone');
```

```
$.post(webViewCallback,JSON.stringify(data),function(data,status){  
  if(status == "success"){  
    console.info("success");  
  }  
  else {  
    console.error(" error status: " + status);  
  }  
});
```

```
 ${webviewresponse.value.origin}  
 ${webviewresponse.value.destination}  
 ${webviewresponse.value.departureDate?numberToDate},  
 ${webviewresponse.value.returnDate?numberToDate}"
```

When registering web applications in a skill, bot designers cannot see the data structure returned by the app.



Debugging using the conversation tester

ORACLE® Oracle Travel

Please edit your information
Complete this form and press submit, or press cancel.

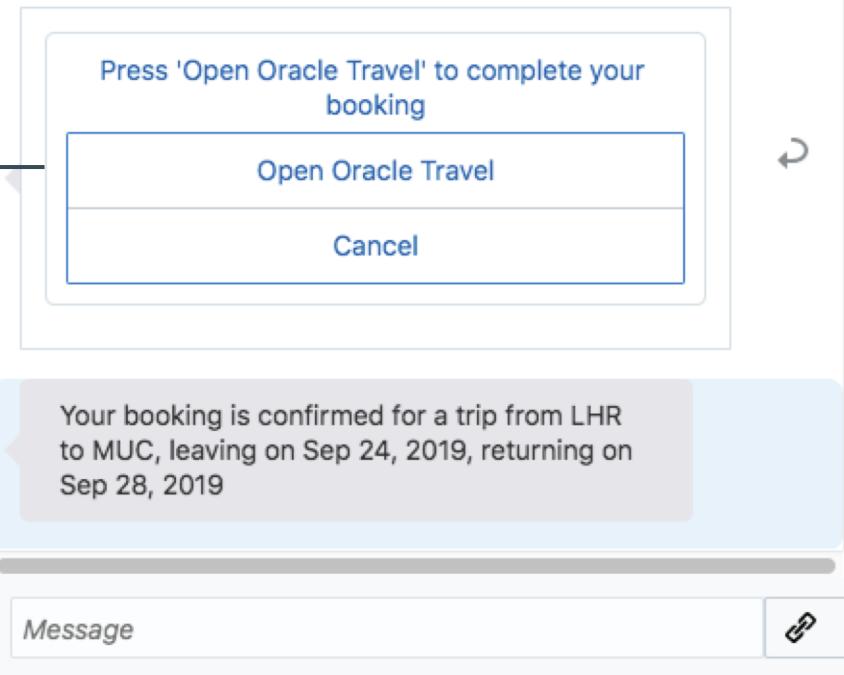
Select Origin
London

Departure Date
09/24/19

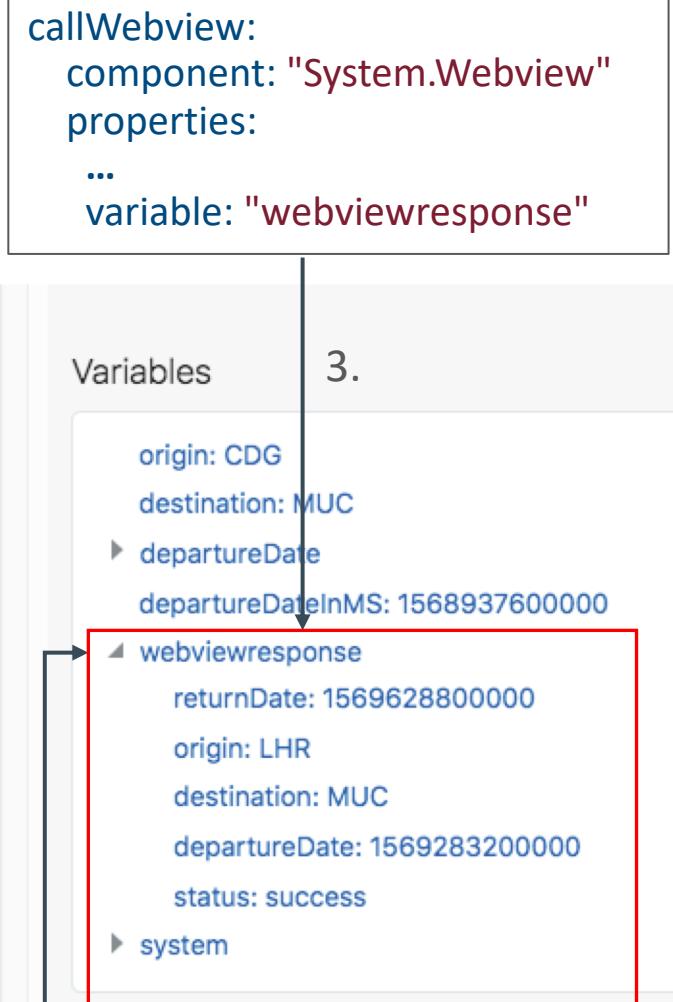
Return Date
09/28/19

Select Destination
Munich

Submit **Cancel**



2.

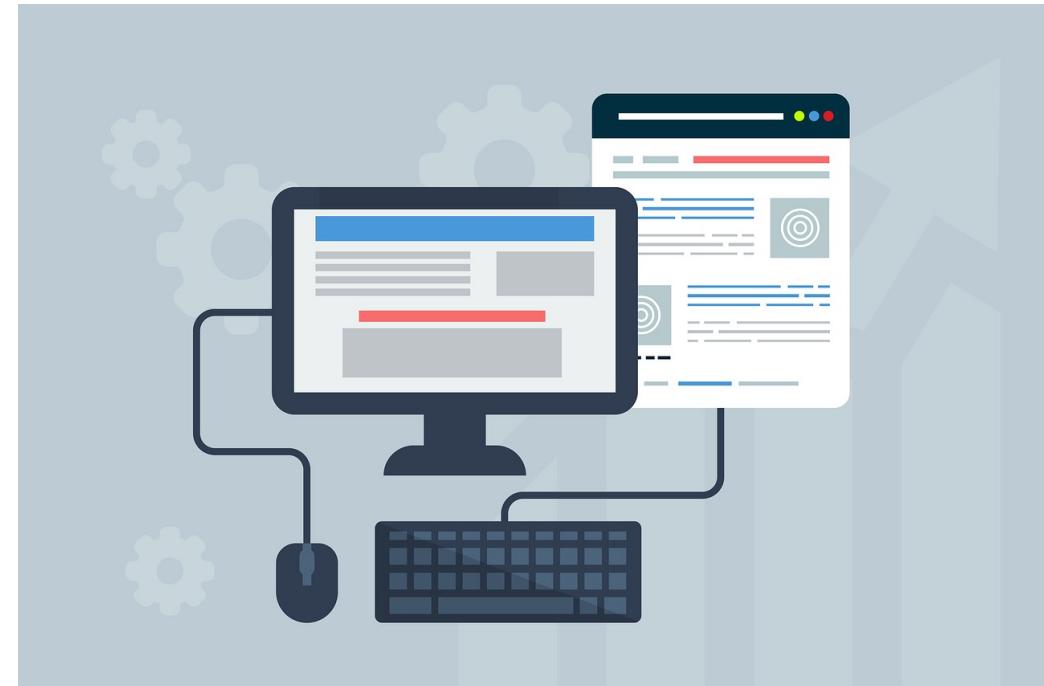


Topic agenda

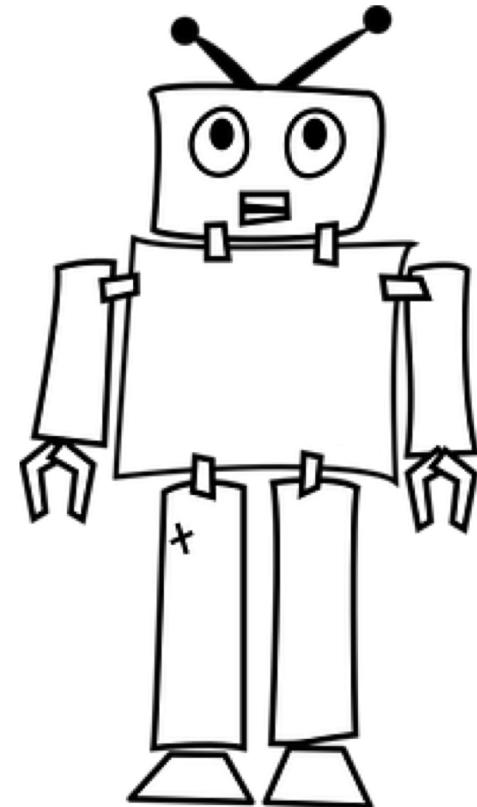
- 1 ➤ When conversation is not enough
- 2 ➤ System.Webview
- 3 ➤ Local web applications
- 4 ➤ Remote web applications
- 5 ➤ Remote web app vs. local web app

Remote web applications

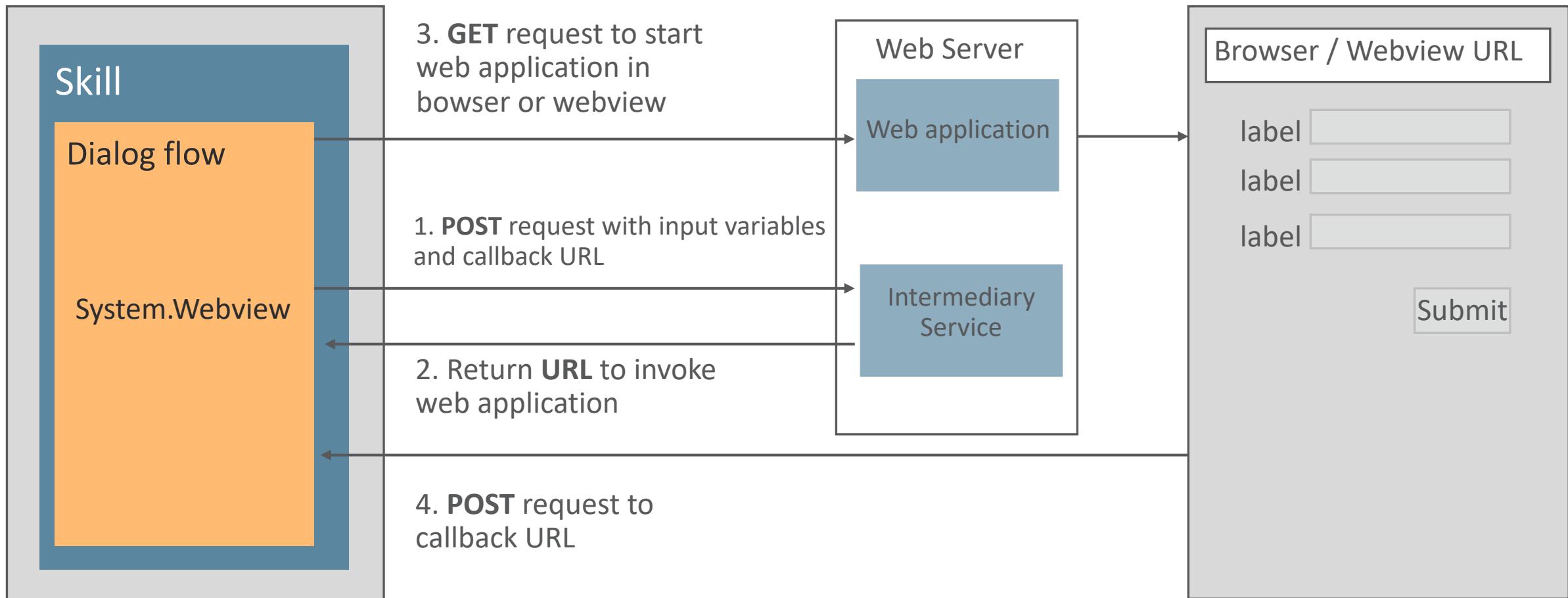
- Existing enterprise web applications
 - Leverage functionality of existing web applications
 - E.g. payment, approvals
- Applications specifically built for use with chatbots
 - Remote deployment because of
 - Data integration requirements
 - Security requirements
 - Central administration requirements



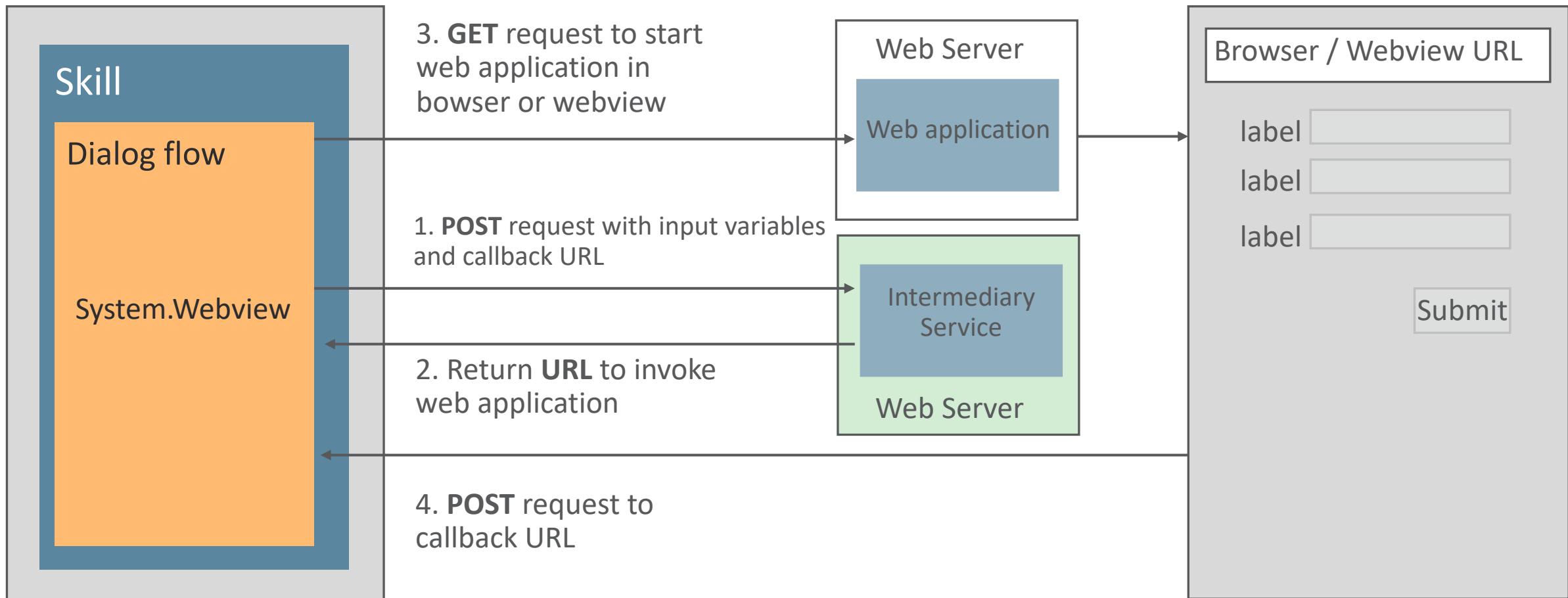
Remote web applications are launched through an intermediary service that prepares the web application request.



Remote invocation architecture



Alternative remote invocation architecture



Intermediary service configuration

Skill

Dialog Flow

```
callWebview:  
  component: "System.Webview"  
  properties:  
    service: "oracletravelweb"  
    sourceVariableList: "origin,destination"  
    variable: "webviewresponse"  
    prompt: "Press 'Open Oracle Travel' ..."  
    linkLabel: "Open Oracle Travel"  
    cancelLabel: "Cancel"  
  transitions:  
    next: "handleResponse"  
  actions:  
    textReceived: "onCancel"  
    cancel: "onCancel"
```

Create Service

Name: oracletravelweb

Description: Optional short description for this service

Service Hosted:

Web App Url: <http://test.com/webapp>

Auth Token: Auth token for user

Query Parameters: JSON object with param value pairs

Create

Intermediary service

- Prepares web application access
 - Creates web application access URL with all request parameters required
 - Chatbot parameters
 - Application specific parameters
- Service can be on same web server or different web server than the web application
 - If on the same server, may be able to save bot request parameters into session, thus avoiding issues with long URL
 - E.g. Internet Explorer supports 2,048 characters, minus the number of characters in the actual path
 - If on different servers, all web application request parameters need to be encoded as query parameters to the web application redirect URL sent to the bot

Payload sent to intermediary service

- POST call with JSON object containing key-value pairs
 - Keys are the names of parameters referenced in the `System.Webview.sourceVariableList` property and "webview.onDone"

```
{  
  "parameters": [  
    {"value": "CDG",  
     "key": "origin"},  
    {"value": "MUC",  
     "key": "destination"},  
    {"value": "https://<url>:443/connectors/v1/callback?state= cb5443 ... 2c ",  
     "key": "webview.onDone"}]
```

Payload of Travel Example

Intermediary service response payload

- Response from the intermediary service must be a JSON payload
- Payload contains single property **webview.url** with URL to remote web application
- Service developer defines the **webview.url** content
 - Can also change parameter names

Service response of Travel Example

```
{  
  "webview.url": "https://<app url>?callbackUrl=https://botphx1.botmfp.ocp.oraclecloud.com:443/  
  connectors/v1/callback?state=7648f4288b&origin=CDG&destination=MUC  
}
```

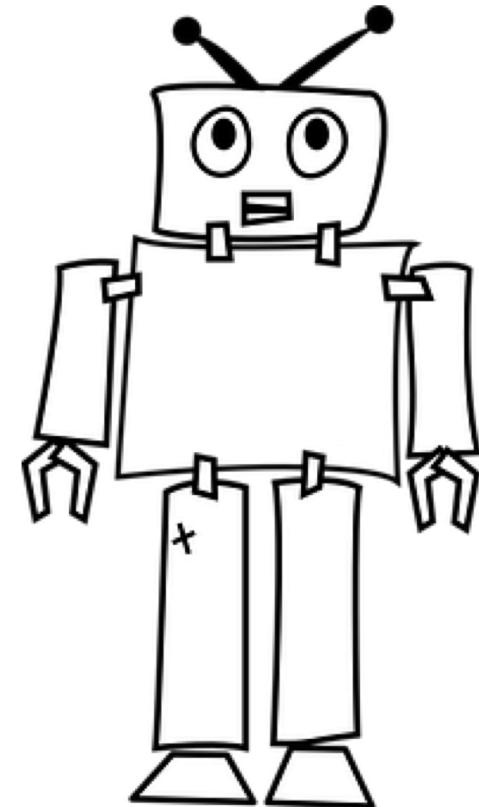
Web application response

- Web application invokes callback URL sent with the initial request to return control to the bot and to optionally send data as a JSON payload

```
let data = {};
data.origin = self.origin();
data.destination = self.destination();
data.departureDate =
  (new Date(self.departureDate())).getTime();
data.returnDate =
  (new Date(self.returnDate())).getTime();
$.post(webViewCallback, JSON.stringify(data));
```

```
{
  "origin": "CDG",
  "destination": "MUC",
  "departureDate": 1568937600000,
  "returnDate": 1568937600000
}
```

Property names in the web application response payload don't need to match variable names in the dialog flow.



Topic agenda

- 1 ➤ When conversation is not enough
- 2 ➤ System.Webview
- 3 ➤ Local web applications
- 4 ➤ Remote web applications
- 5 ➤ Remote web app vs. local web app

Remote vs. local web application

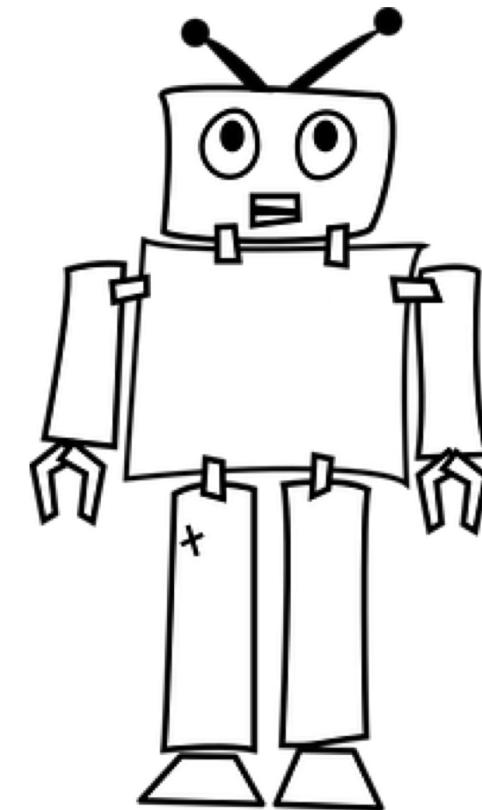
Remote web application

- Application needs to be modified or created to work with webview
- Good for Integrating existing web applications or functionality
- Security through web application authentication
- Leverage server side infrastructure
 - E.g. environment variables

Local application

- Application created for deployment to local container
- Easy support for structured data input and missing widgets (e.g. date picker support)
- No application security
- Good to use for self-contained skills deployed via skill store

The limitation of web applications displayed in webviews is that they don't work with voice.



Integrated Cloud Applications & Platform Services

ORACLE®



Oracle Digital Assistant Hands-On