
ORACLE®

Oracle Intelligent Bots Advanced Training

Advanced Custom Component Topics

ORACLE

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.

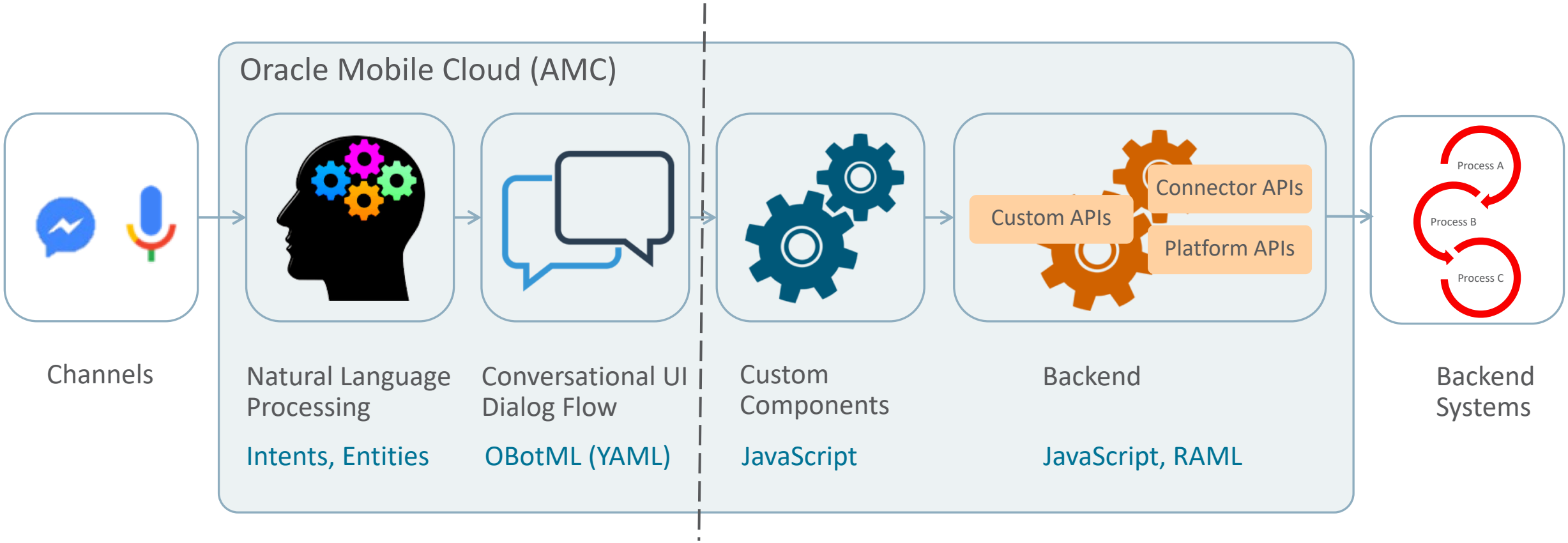
Topic Agenda

1. Custom Component Overview
2. Custom Component SDK
3. Backend Integration
4. Conversation Message Model (CMM)
5. CR Component vs. Custom Component and CMM

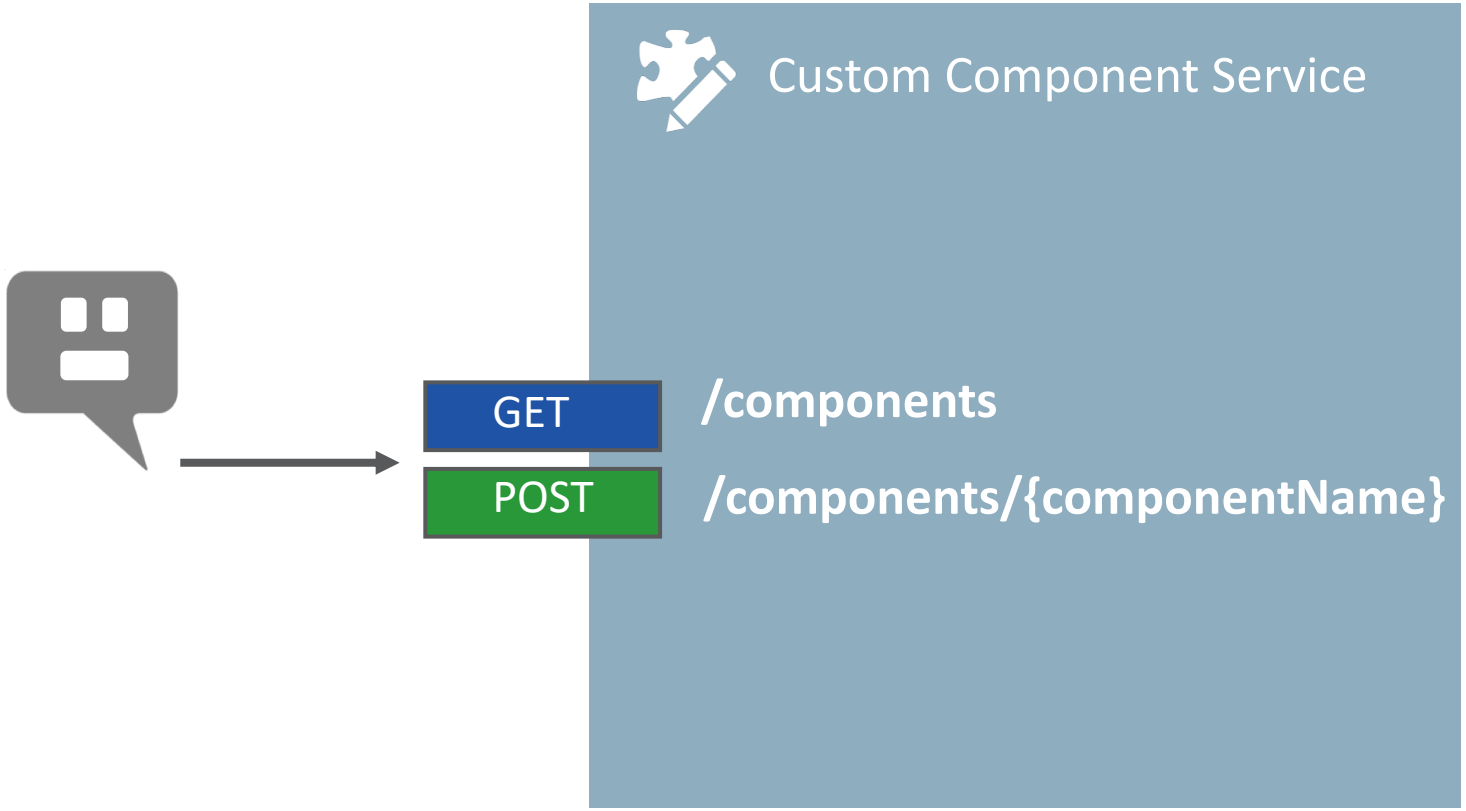
Custom Component

Overview

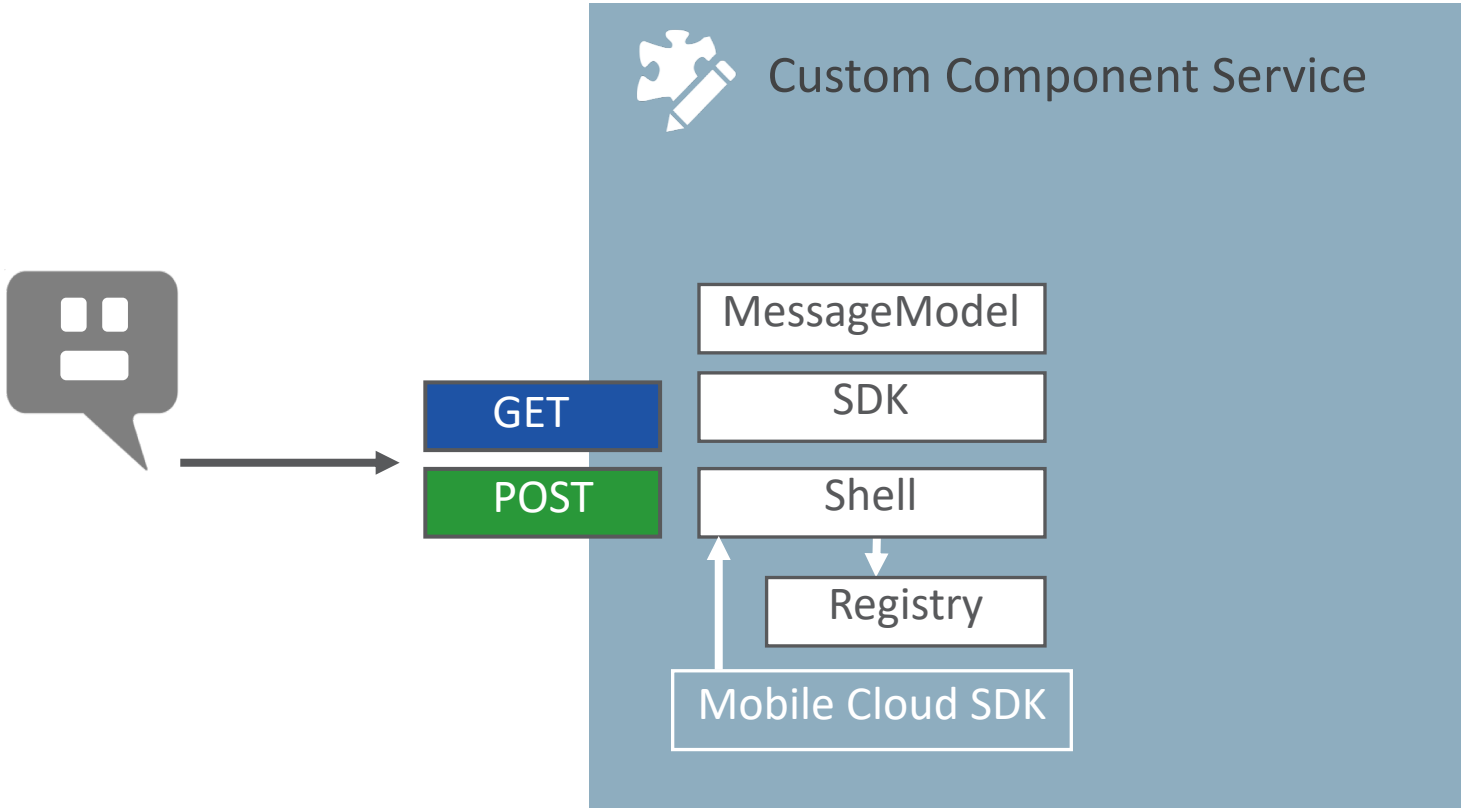
Bots Application Architecture



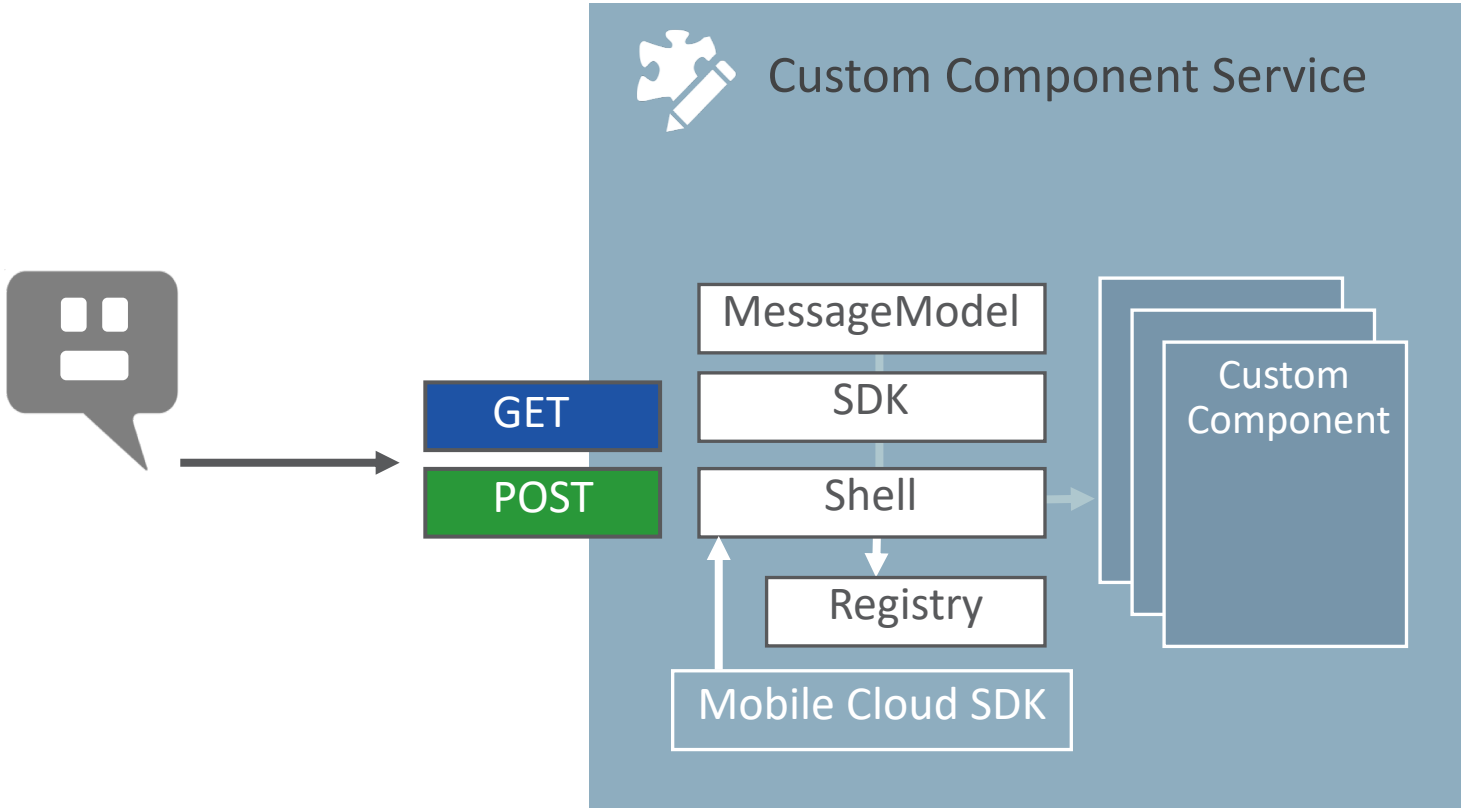
Custom Component Architecture in Mobile Cloud



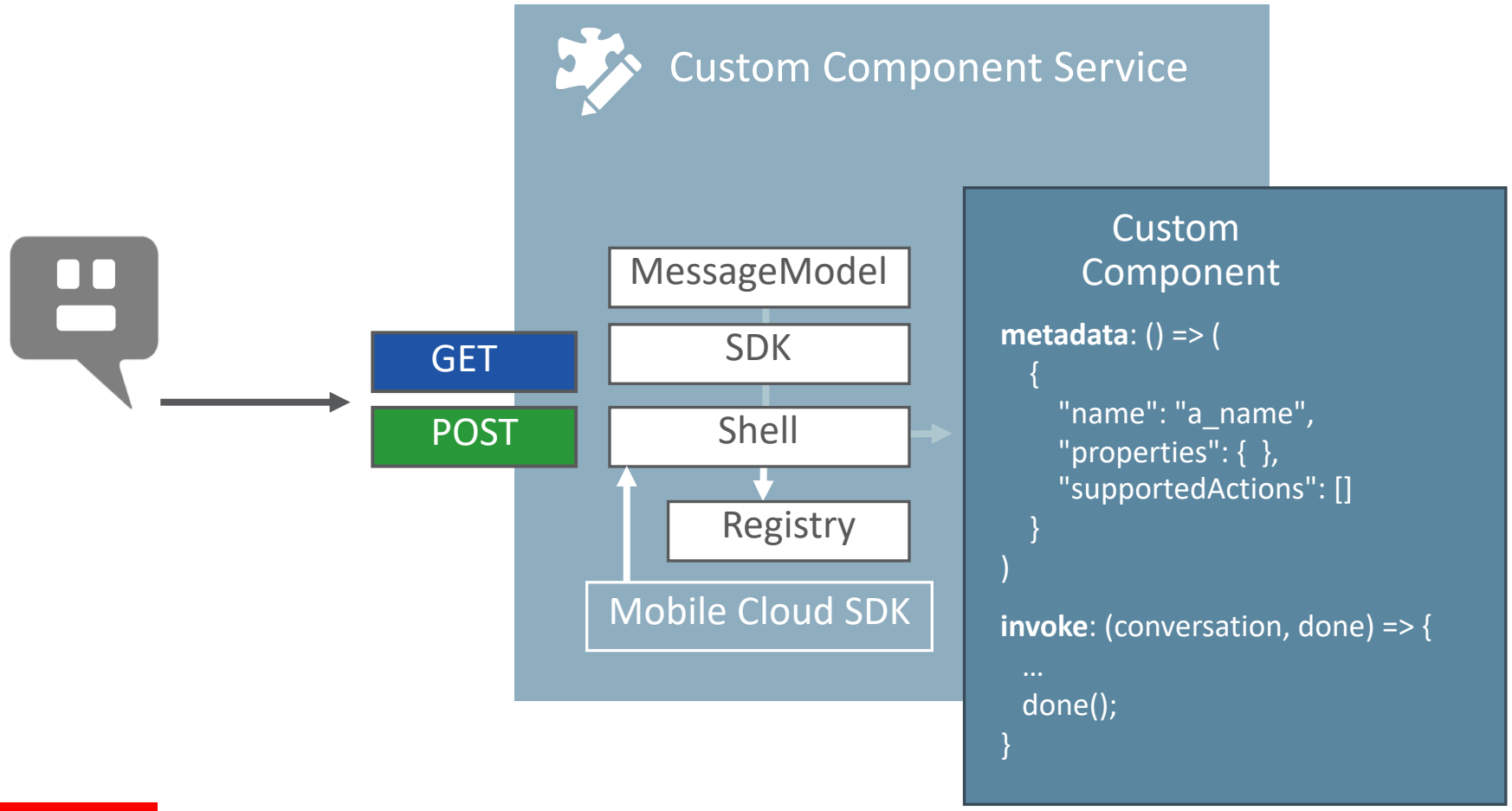
Custom Component Architecture in Mobile Cloud



Custom Component Architecture in Mobile Cloud



Custom Component Architecture in Mobile Cloud



Custom Component

Custom Component SDK

Sending Response Messages to the Bot

conversation. reply(…)	• Returns message directly to user • Arguments can be of string, object and MessageModel type • Can be called multiple times for a single response • Sets keepTurn to false
conversation. reply(String)	• Sends a simple text message in a bubble • No formatting. Just plain text.
conversation. reply(ConversationMessage)	• Conversion Message Model message • Supports card - , attachement -, location - , postBack -, raw responses

Accessing Bot Messages - Text

conversation.rawPayload()	{"messagePayload":{"**text**":"hello world","type":"text"},"profile":{"firstName":"john","lastName":"doe","timezoneOffset":-3600000,"locale":"en-US"},"userId":"8547335"}
conversation.messagePayload()	{"type":"**text**","text":"hello world"}
conversation.text()	hello world
conversation. payload()	deprecated: same as rawPayload()

Accessing Bot Messages – postback, location, attachment

conversation.postBack()	• Postback messages are sent from action items (e.g. button) • Returns a JSON object with key value pairs • Returns null if message is not a postback
conversation.location()	• Returns JSON object with longitude and latitude information • Returns null if message is not a location message
conversation.attachment()	• Users may use the CR component to upload image, video, files and audio content. The url of the attachment content is sent to the component. • Returns JSON object with the url of a user provided attachment • Returns null if message is not an attachment message

Working with Variables and Properties

conversation.variable('variable_name')	• Reads the value from a named contex variable
conversation.variable('variable_name', value)	• Updates a named context variable • If the variable doesn't exist, it will be created
conversation. properties().property_name	• Reads value from a input parameter

Context variables that are created by a custom component cannot be referenced from BotML expressions.

Create context variables at design time and use a custom component to populate the data if BotML access Is a requirement.

Passing Variable to Custom Component

callCustomComponent:
component: "custom.component.name"
properties:
dataVariable: "add_name_of_context_variable_to_hold_data"

```
metadata: () => ({
  …
  "properties": {"dataVariable": {"type": "string", "required": true}},
  …
})

invoke: (conversation, done) =>{
  const resultVariable = conversation.properties().dataVariable;
  …
  conversation.variable(resultVariable, value);
  … }
```

Accessing Intents and Entities from NLP Processing

var nlpresult = conversation.nlpResult(string)	• Accesses the nlpresult instance for the nlpresult variable referenced in the argument (e.g. iResult) • Argument can be empty if only a single nlpresult variable is used in a dialog flow.
nlpresult.entityMatches(string)	• Returns an array of entities extracted from the user string that match the provided entity name • If no value is provided then all entities found in the user string are returned
nlpresult.intentMatches()	• Returns array of intents resolved for the processed user string • Intent is a JSON object with an "intent" and a "score" property
nlpresult.topIntentMatch	• Returns the top resolved intent as a JSON object with an "intent" and score" property

Example

```
invoke: (conversation, done) => {
  const nlpresultVar =
              conversation.properties().nlpresultVariable;
  const entityName = conversation.properties().entityName;

  //get nlp instance for provided nlp variable
  var nlp = conversation.nlpresult(nlpresultVar);

  var entities = nlp.entityMatches(entityName);
  …
}
```

Trigger Dialog Flow Navigation

conversation.transition(string)

- Custom components may define supported action strings in their metadata that dialog flow designers use to determine the next state to navigate to
- Custom component developers use the transition(...) function to return an action string
- Its up to the dialog flow designer to use the action string for navigation or to ignore it

Request User Input

conversation.keepTurn(true)	• Used by components that need to print a message to the user but don't require the user to provide an input in response • Use case would be to print a statement
conversation.keepTurn(false)	• Most commonly used • User is required to provide an input in response to a component response • Use case would be a list of value printed by the component that the user should select a vaue from

The **combination of keepTurn**(boolean) and **transition**(string) **determines** the **when and how** the **dialog flow navigates to a next state**

Image courtesy of Farbentek at FreeDigitalPhotos.net

conversation.reply() **implicitly** sets keepTurn(false)

If needed, set keepTurn(true) after conversation.reply()

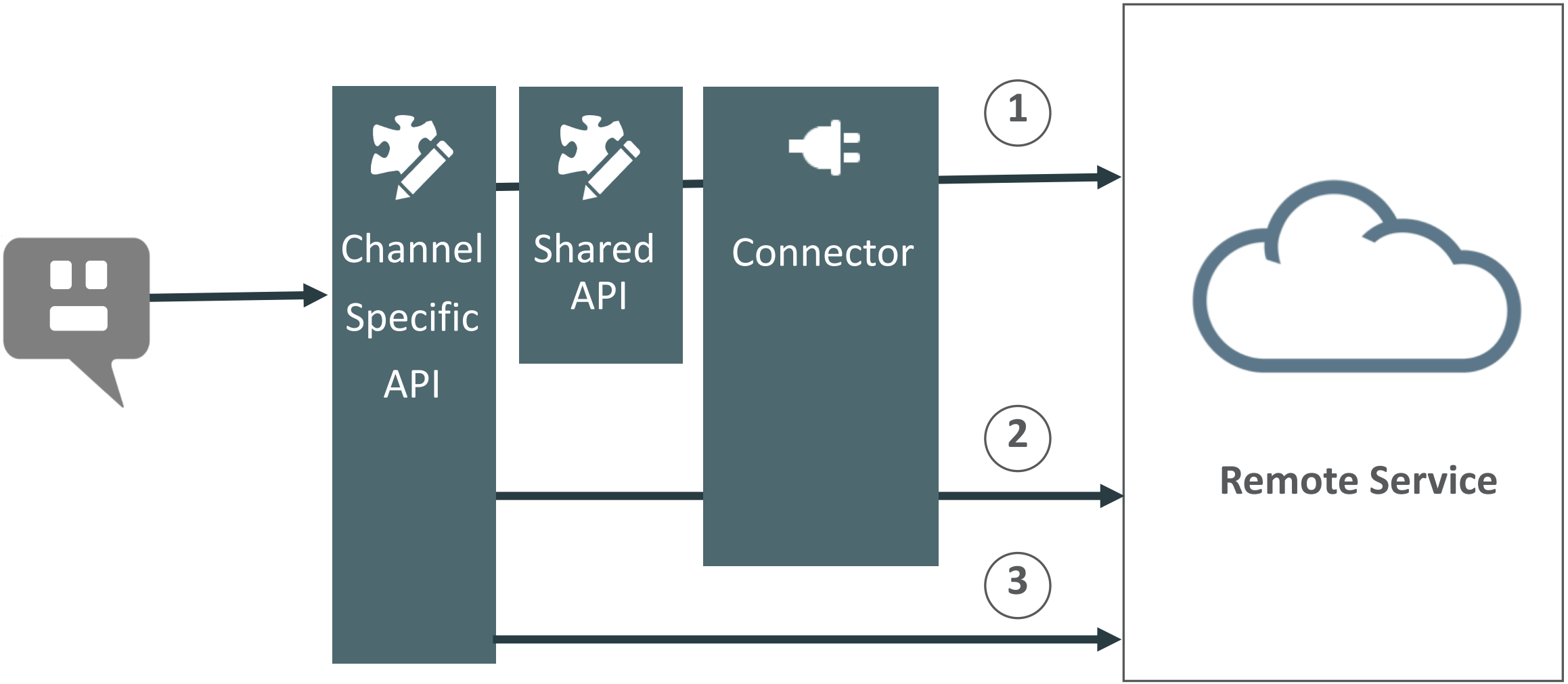
conversation.logger().<level>(message)

- Shared logger defined by the shell object
  - Log levels depend on logger that is used
- Console logger is default
- Oracle Mobile Cloud console log levels
  - Severe
  - warning
  - Info
  - Config,
  - Fine, finer, finest
- Logs in Oracle Mobile Cloud are displayed in diagnostics panel of MBE

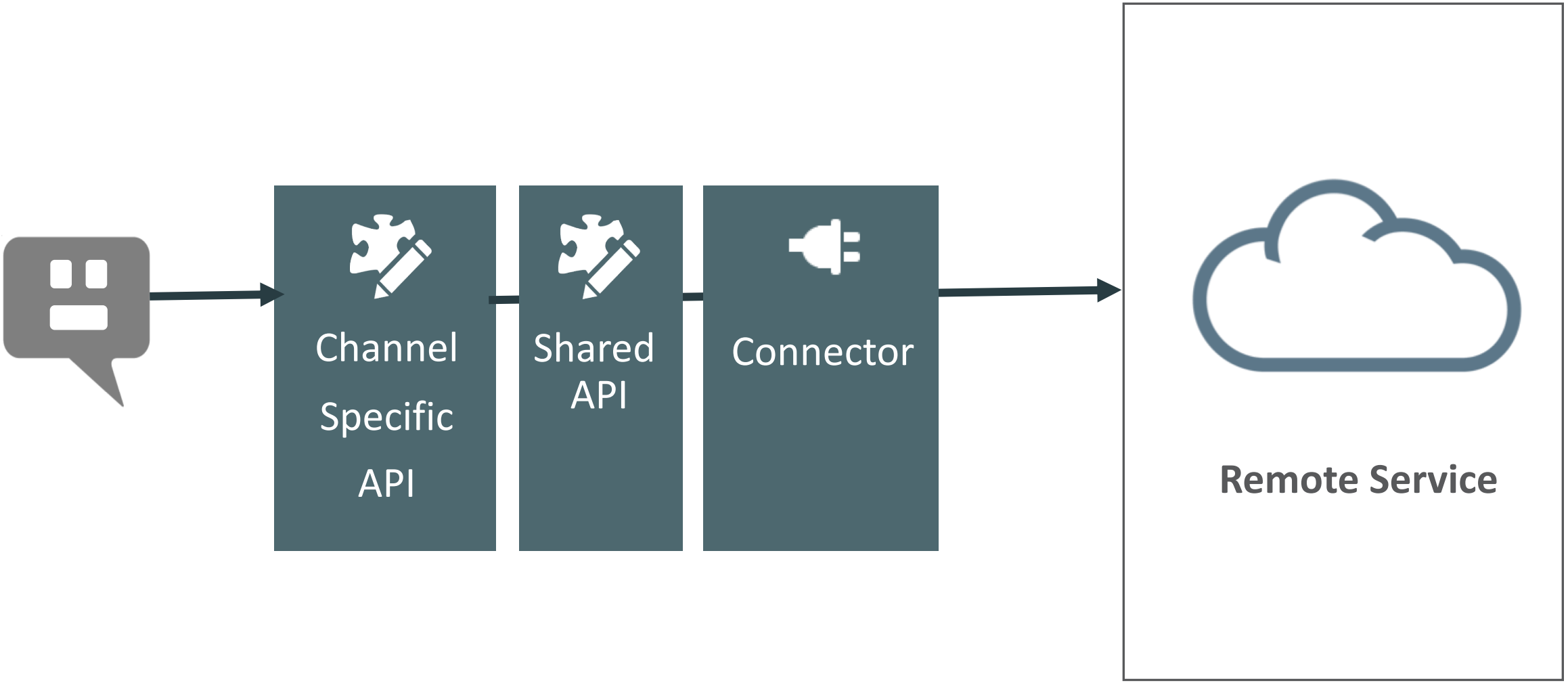
Custom Component

Backend Integration

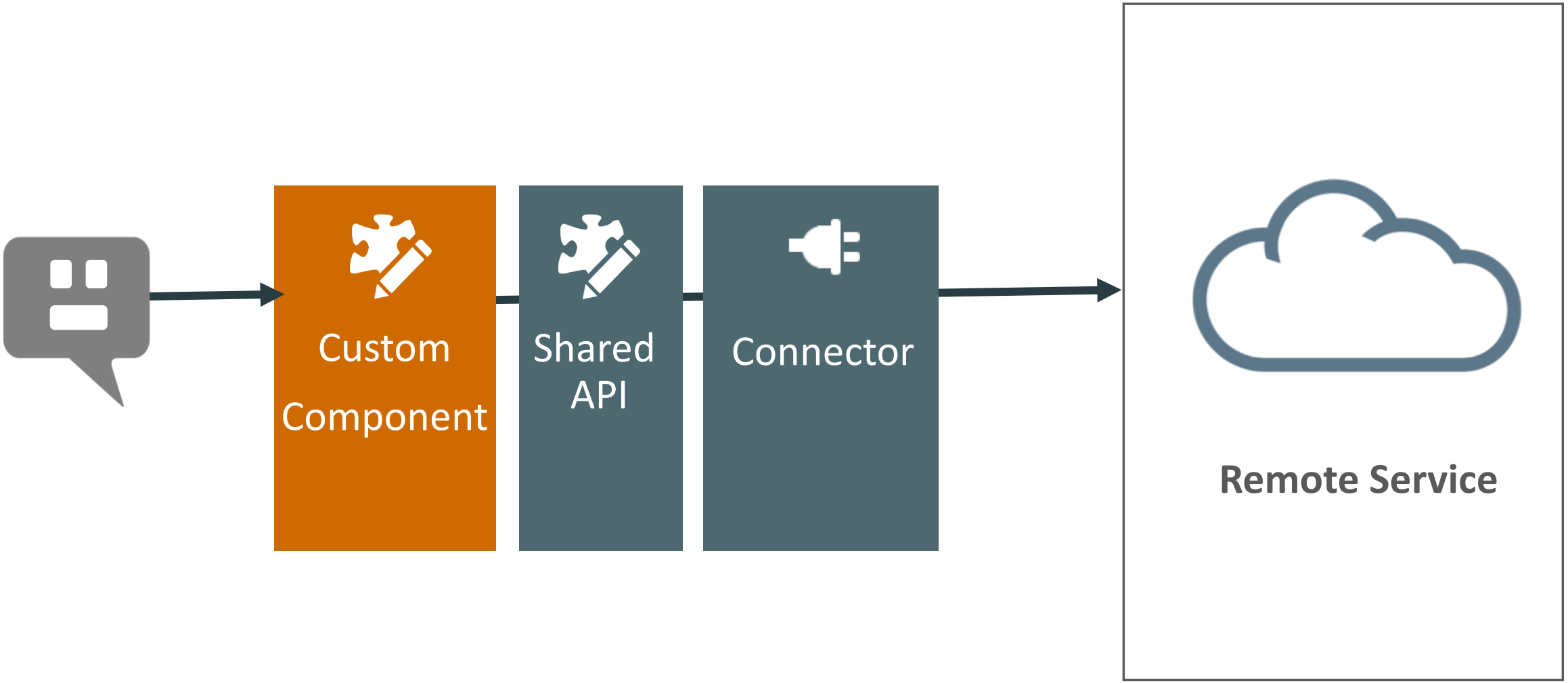
Oracle Mobile Cloud Backend Integration



Oracle Mobile Cloud Multi-Channel Backend Integration



Oracle Mobile Cloud Bot Backend Integration



Advanced Bot Training Hands-On

Lab 5b: Building a menu from data provided by a custom component

Custom Component

Conversation Message Model

User and bot messages share a **single message model**, the Conversation Message Model (CMM)

Image courtesy of Farbentek at FreeDigitalPhotos.net

Conversation Message Model (CMM)

- Consistent Messages Structure Format
- Use of Connectors to Produce Channel Specific Message Responses
- Webhook returns CMM Message Structure as is
- Design-time Support
  - Common Response Component
  - Custom Component (SDK)

Message Payload Types

- Text
  - Sending text messages
- Post back
  - Text or JSON object payload
- Location
  - Send longitude and latitude information as a message
- Attachment
  - Video, File, Audio, Image
  - Attachment URL sent as the payload

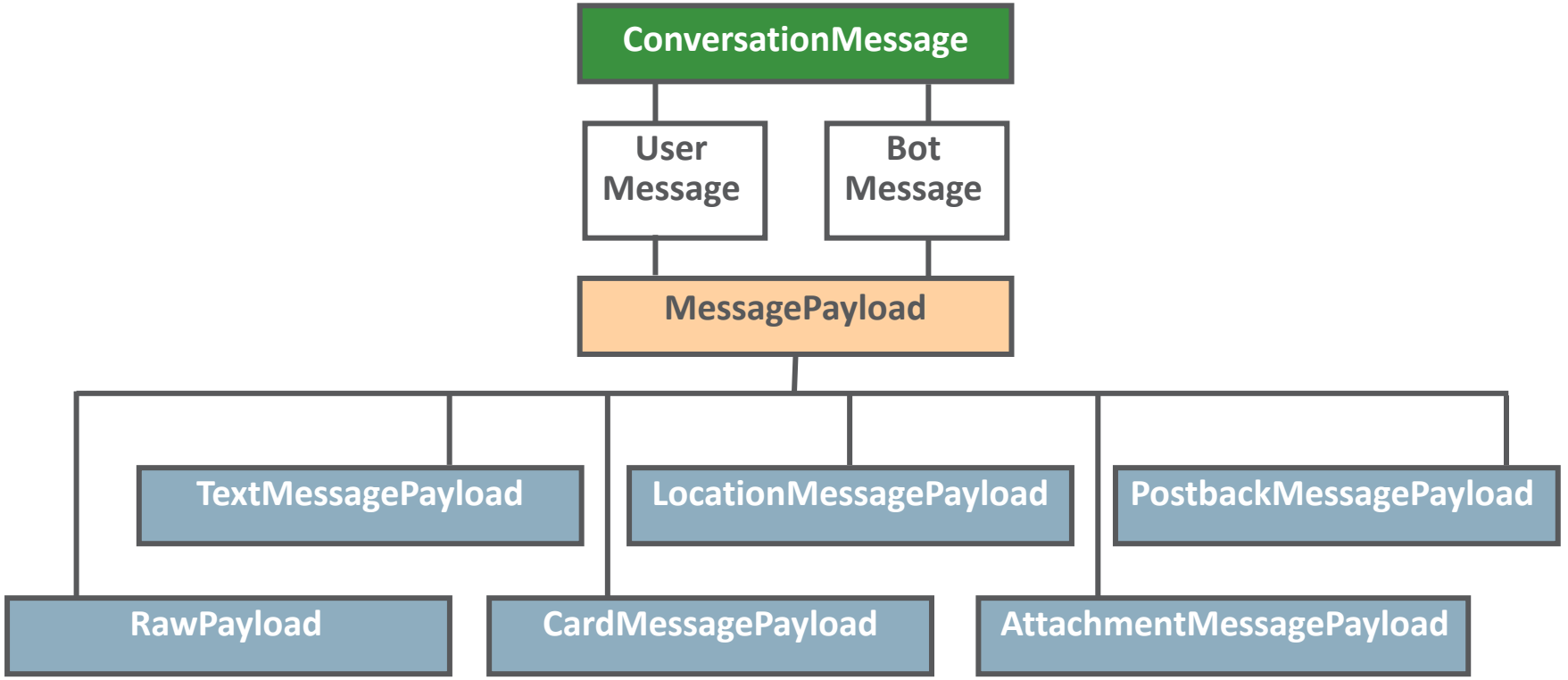
Message Payload Types

- Cards
  - Displays a choice as vertical or horizontal cards
  - Requires messengers that support cards
- Raw
  - Allows channel specific messages to be sent
  - E.g. using messenger specific templates

CMM Message Hierarchy

Shared model

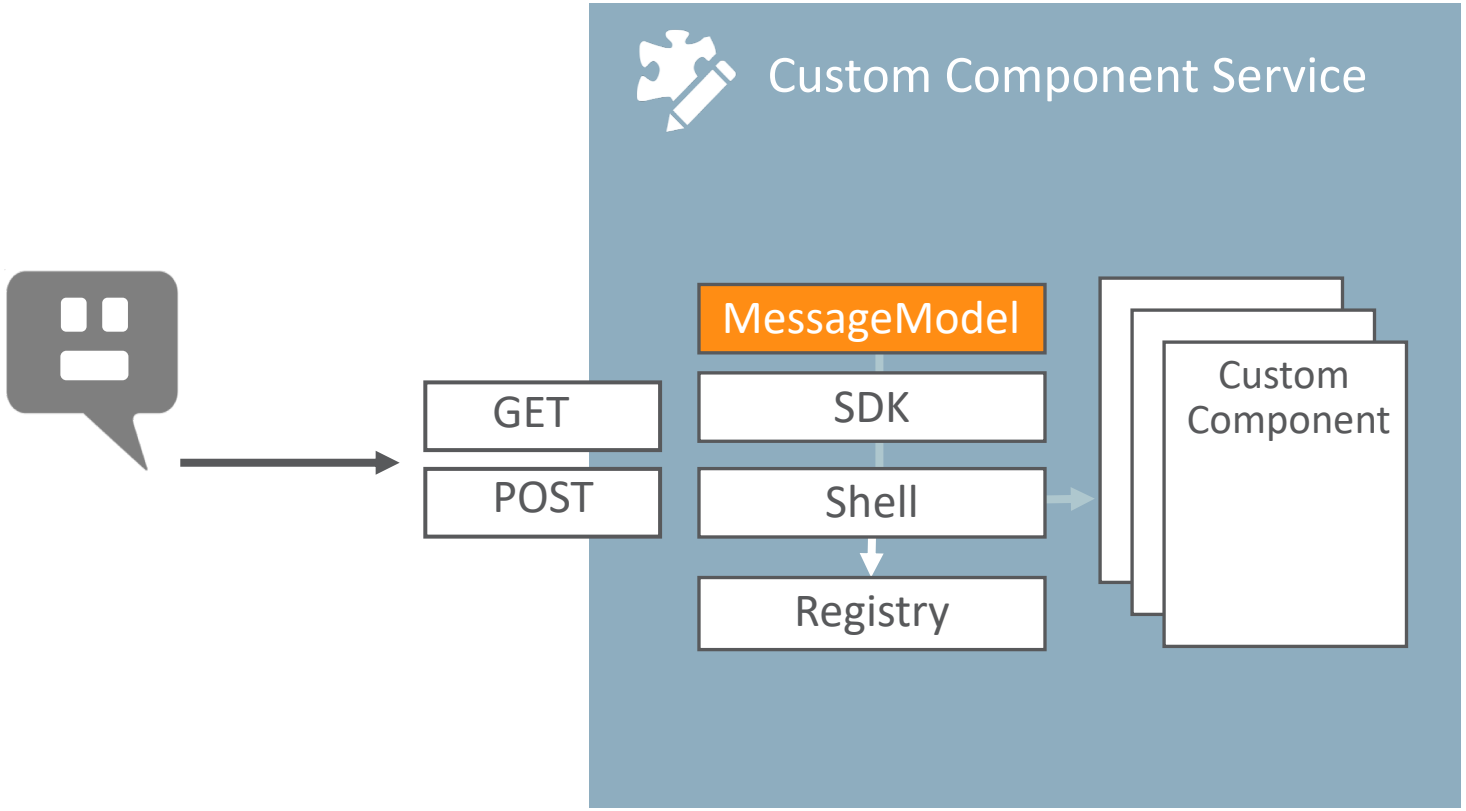
Message Type Specific



The custom component SDK's **MessageModel** class provides shortcut functions for creating CMM bot responses

Image courtesy of Farbentek at FreeDigitalPhotos.net

Message Model Class in the SDK



MessageModel

- Bot SDK Utility class
  - Creates and validates bot messages
  - Exposes helper functions to access messages
    - CMM and RAW format
- Accessed through SDK
  - `conversation.MessageModel().< function >`
- Exposes function for creating messages of type
  - *Text, Card, Attachment, Location, Postback, Raw*

TextConversationMessage

- Writes a text message to the messenger client
  - Simple text, JSON object
- Optionally, takes an array of actions that are rendered below text
  - postbackActionObject, locationActionObject, urlActionObject, shareActionObject

```
var messageModel = conversation.MessageModel();

var payload = 'hello world';
var actions = [];
…
var textResponse = messageModel.textConversationMessage(payload, actions);
conversation.reply(textResponse );
```

CardConversationMessage

- Shows responses as an array of vertical or horizontal cards
- Optionally, takes an array of actions that are rendered on each card
  - postbackActionObject, locationActionObject, urlActionObject, shareActionObject

```
var messageModel = conversation.MessageModel();

var cardObject = messageModel.cardObject("title string", "description",
"imageUrl", "card Url", [array of actions]);

var cards = [];
cards.push(cardObject);

var cardsResponse = messageModel.cardConversationMessage('vertical', cards);
conversation.reply(cardsResponse );
```

AttachmentConversationMessage

- Creates image, video, audio, file response
- Each attachment is rendered in its own bubble

```
var messageModel = conversation.MessageModel();

var type = "image" //video, audio, file
Var docUrl = "http://host:port/my_image.jpg";

var attachmentResponse = messageModel.attachmentConversationMessage(type,docUrl)

conversation.reply(attachmentResponse );
…
```

There are three more:

`locationConversationMessage`,
`postbackResponseMessage`
`rawConversationMessage`

return JSON payloads and require a custom messenger client. Otherwise you don't use them in a custom component.

An **action is something a user can select**; a list item or button

CMM ActionObject Overview I/II

- Postback action
  - Individual user payload sent back to the bot upon selection
  - `var action = messageModel.postbackActionObject( label, imageURL, payload)`
- Call action
  - A phone number to dial on a mobile device
  - `var action = messageModel.callActionObject( label, imageUrl, phoneNumber)`

CMM ActionObject Overview II/II

- URL action
  - A web address to open in a mobile browser
  - `var action = messageModel.urlActionObject(label, imageUrl, url);`
- Share action
  - Ability to share messages in a messenger
  - `var action = messageModel.shareActionObject(label, imageUrl);`
- Location action
  - Ability to request and receive user GPS location
  - `var action = messageModel.locationActionObject(label, imageUrl);`

Custom Component

Handling User Input

"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 Farbentek at FreeDigitalPhotos.net

For a component to also handle the user response you need to pause state transition

Pausing State Transition to Handle User Responses

- Feature of the CR component
  - "processUserMessage"
- To implement the same
  - Initial component request
    - **No** call to `conversation.transition()`
    - keepTurn = **false**
    - Call `done()` at end of component processing
  - Subsequent request
    - Call `conversation.transition()`
    - keepTurn = true | false
    - Call `done()` at end of component processing

Custom components that handle user responses must handle their state

Image courtesy of Farbentek at FreeDigitalPhotos.net

Custom Component State Handling

- Component Managed Context Variable
  - Variable created at runtime
- Component Action Payload
  - Add property to e.g. postback payload

```
var _changeAction = conversation.MessageModel().postbackActionObject(_changeLabel, null, {
    "orderAction": "change",
    "orderId": _orderId,
    "statetoken": "advt24hrsflowers_fn_1234567"
});
```

Determine the User Input Message Type

- `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 ..}

Example: Setting Variable Value From Postback Payload

variables:
  pizzaType: "string"
  pizzaSize: "string"

postback payload:

```
{
  "pizza": "Supreme",
  "size": "large"
}
```

```
invoke: (conversation, done) => {
  //handle postback response
  if (conversation.postback()) {
    //postback contains a key-value object
    let postbackpayload = conversation.postback();
    //update variable with a postback value
    conversation.variable("pizzaType",
                          postbackpayload["pizza"]);
    conversation.variable("pizzaSize",
                          postbackpayload["size"]);
    conversation.transition();
    done();
  }
```

There is a design flow in the code below, **what is it?**

```
invoke: (conversation, done) => {
  //handle postback response
  if (conversation.postback()) {
    //postback contains a key-value object
    let postbackpayload = conversation.postback();
    //update variable with a postback value
    conversation.variable("pizzaType",
                          postbackpayload["pizza"]);
    conversation.variable("pizzaSize",
                          postbackpayload["size"]);
    conversation.transition();
    done();
  }
```

Custom Component

CR Component vs. Custom Component and CMM

CRC Extra Powers

"On-board" features of the CR component you need to code using CMM in a Custom Component

- CR component features
  - rangeStart, rangeSize (only cards)
  - unexpectedAction
  - processUserMessage
  - Composite responses
  - Support for composite entity bags
- CR functionality
  - Entity slotting

- Component properties
  - keepTurn
  - variable
  - nlpResultVariable
  - translate
  - maxPrompts

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 Use CMM

- When the custom component should be a "closed" system
  - Handles a complete task
- When a component needs to go into longer "user interactions"
- When a component should be reused across bots
  - Avoid dialog flow dependencies

Advanced Bot Training Hands-On

Lab 5c: Create a Card Layout Menu Using the CMM

Integrated Cloud

Applications & Platform Services


ORACLE®