Order Buddy

Aanchal Anil Samdariya: samdariya.a@husky.neu.edu
Apoorva Kasoju: kasoju.a@husky.neu.edu
Karan Gulur Muralidhar: gulurmuralidhar.k@husky.neu.edu
Krishna Sai Rohith Kanuparti: kanuparti.k@husky.neu.edu

December 7, 2019

I. Summary

Traditional way of placing an order involves repetitive manual labour, be it giving/taking orders face to face at an eatery or using mobile/web applications for ordering. Leveraging increase in popularity of hands free assistant[1] such as Google Home and Alexa, we enable the conversational agent to work as a Speech-to-text Order Assistant.
The user is given the ability to navigate through all phases of order management (i.e., placing an order along with customization, modifying the existing order and cancel an item/order), in effect, have a conversation and be able to place an order hands free.
The following could be an example of the kind of conversation possible with Order Buddy:

OB: Hey there! What do you want to order?
Cust: I want a Mocha Latte with some whipped cream.
OB: What size?
Cust: Large
OB: Got it, one large Mocha Latte with whipped cream. Do you want to order anything else?
Cust: One small Coffee with soy milk please.
OB: Got it, one small Coffee with soy milk. Do you want to order anything else?
Cust: No, that's it.
OB: Order for 1 large Mocha Latte with whipped cream and 1 small Coffee with soy milk is confirmed, have a good day!

The history of all orders are stored for future reference and the entire application is integrated with the Google Assistant.
A subset of the menu from Starbucks will be used as the dataset for proof of concept and later aim to expand to any menu.

II. Methods

On a high level, we have the user who is facing the Google voice assistant. Any utterance emitted by the user to the voice assistant will be converted to text using Google Speech to Text APIs[2]. The text form of the utterance is then forwarded to the chat bot framework (implemented with Dialogflow[5]) which is responsible for intent matching and entity extraction. The different components implemented using Dialogflow can be seen in the Appendix. The request is then forwarded to the back-end code which is responsible for business logic and data storage (Google Cloud Firestore[6]) and the response is returned back to the user.

To better explain the different aspects of this project the following are the broad sections to which we sub divide methods section as follows:

i. Entity Extraction

In order to identify what a particular user is referring to, our chat bot framework should be able to identify the menu items, size customization parameters a user mentions in the utterance. Even though we have the state of the art entity extractor to identify place, location

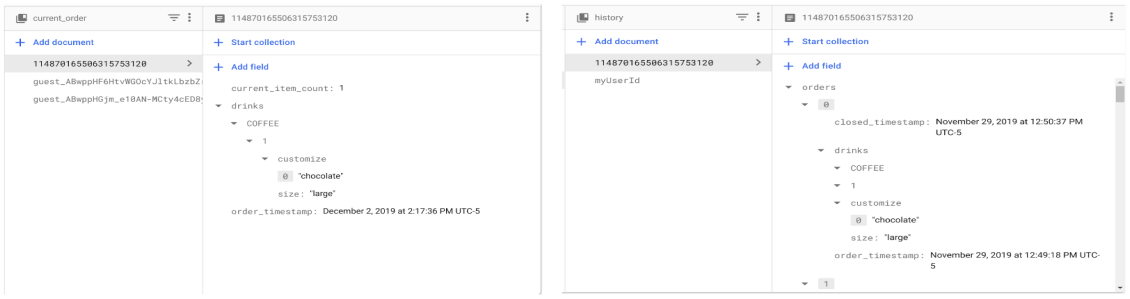

Figure 1: *Firestore example*

and organization words, a custom entity model is to be developed to recognize restaurant specific menu item names. Dialogflow provides us the with framework where we can define new entities with a small probable set of values and also give us the flexibility to list synonyms. For this project, we create 'Drinks', 'Customization' and 'Size' entities in Dialogflow. For Ex, an 'large' is a value mentioned under the entity 'Size' with synonyms 'huge' and 'Venti'. Similarly we have synonyms listed for all the drinks and customize options in Starbucks menu.

ii. Intent Identification and Training

Every user utterance should be analysed mapped to a particular action - order, cancel_item, cancel_order, complete_order. The process of mapping an user utterance to a corresponding action is called as intent identification. Dialogflow provides a framework to build a model to identify intents. It expects to define different intents along with the sample training phrases which encapsulates how user perform a specific action. It takes these inputs and then builds a model in the back-end extracting the sentence structure and generates some more sentences internally to be able to handle a wide variety of utterances incident as input and gives back the trained model which then can be used to identify intents.

We also make sure to diverge the sentence structure required to trigger an intent and restrict the flow of conversation amongst intents to avoid any situation that might lead to a one-to-many intent matching. For more detailed information on number of training phrases used for each intent please refer Results (Section 3).

iii. Google Cloud Firestore

Cloud Firestore[5] is a cloud-hosted, No-SQL database that allows for mobile, web, and server development. The Cloud Firestore is used to handle and store all the customer's current and past orders as seen in Figure 1. To achieve this, we have created two collections:

- Current Orders: Orders which are still under process and not complete are maintained in this collection. Every time an update is made to an order, the changes are reflected in this collection which is made possible through our custom logic based on the type of update made to the order. Here, every current order is stored along with the id of the customer/user.
- History Orders: To keep a record of all the past orders for every customer, we use the 'History Orders' collection which is updated every time a customer completes an order. All the past orders grouped by customer/user id.

iv. Intent Transition

To achieve intent matching, four main intents are implemented which is Order Intent (to add an item in the cart), Cancel Item Intent (to cancel a previously ordered item), Cancel Order intent (to cancel the entire order) and Complete Order Intent (to complete/place an order).

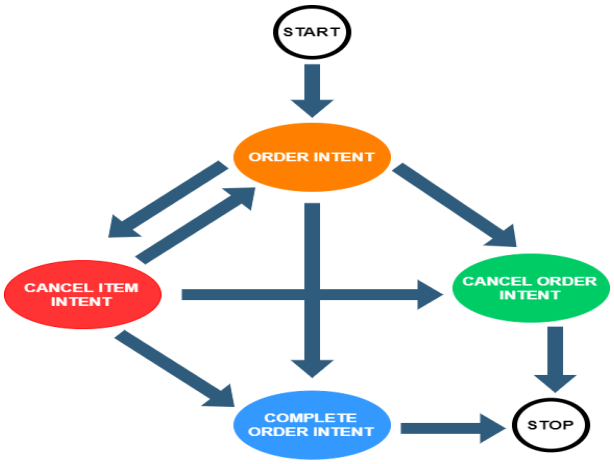

Figure 2: *State Transition Diagram*

The transition between intents can be seen in Figure 2. Any intent can be called at any point of the conversation depending on the user's utterance. Usually, a customer would start with placing an order. Once an order is placed, the customer can proceed to addition of another item, cancellation or completion of an order. Likewise, once a customer removes an item from the order, any of the other intents can be called based on the user's choice.
Once the customer, completes/ cancels the entire order, it would denote the end of the conversation. Now, OrderBuddy has to be invoked once again to start a new order.

v. High Level Architecture & Working

The following steps explain in detail the transition of states and steps taken on an end-to-end basis as seen in Figure 3, starting from users utterance, to reply from 'OrderBuddy' through Google Messenger. These steps correspond to a conversation of a user trying to place an order of *'Large White Chocolate Mocha with Almond milk and Cinnamon'*

- User converses with a personal Google Home device and triggers 'OrderBuddy' by requesting to switch to the agent.
- After switching to OrderBuddy a check is made on user token to determine if the user has already granted permissions to access his Google account.
- User says *'I want a White Chocolate Mocha'*. This utterance is transcribed and processed by Dialogflow. with following parameters extracted:
  - Intent: Order Intent
  - Drink: White Chocolate Mocha
  - Size: Null
  - Customization: Null
- As we have designed 'Drink', 'Size' and 'Customization' as mandatory parameters for ordering an item, we expect them as part of the input. Dialogflow follows up with what is called 'Slot-filling' by proceeding to ask questions until all the mandatory parameter options are obtained, resulting in replies *'What is the size?'* and *'Please specify your customization, if not please say no'*.
- User responds to these questions with *'Large'* and *'Almond milk, Cinnamon'* correspondingly and now the parameters obtained by DialogFlow are as follows:
  - Intent: Order Intent
  - Drink: White Chocolate Mocha
  - Size: Large
  - Customization: [Almond milk, Cinnamon]
- With all the parameters for a Order Intent present DialogFlow sends the intent and

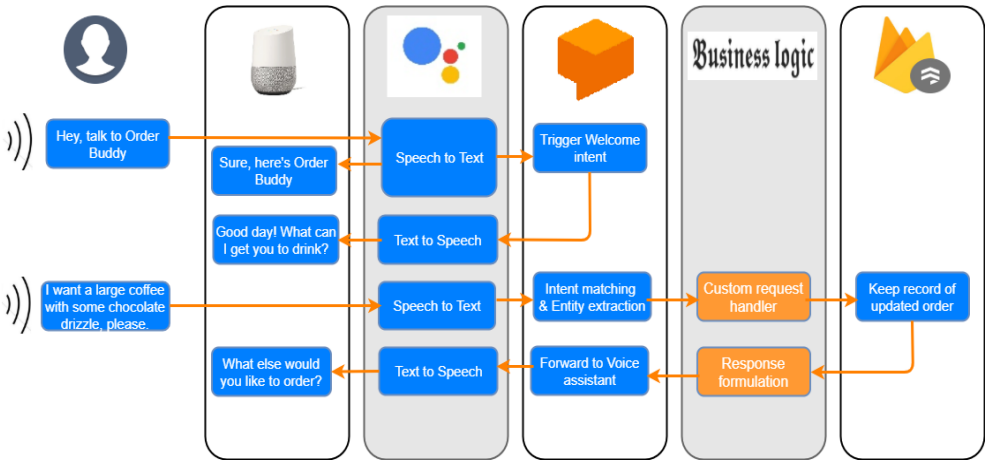

Figure 3: *High level architecture*

parameters amongst other details as a Fulfillment request to our back end server maintained by 'Flask' which uses Rest API to listen to requests.

- The back-end server receives this request and identifies the intent and calls the corresponding custom function which handles requests corresponding to Order Intent.
- The custom function then initially performs a check to test the null status of document in our database (Google's Cloud Firestore) and creates one for the user if no document is detected. After a document exists, we proceed to add the current drink with its parameters to the nested dictionary structure 'Drinks' under key 'White Chocolate Mocha' with an 'Item Number'.
- Custom function also assigns an integer 'Item Number' to the drink added as described previously, which is used to uniquely identify this item for this particular user in the current ongoing order of the user. (This number is used in other intents such as 'Cancel-Item intent')
- The custom function then responds with a 'Success' and also the *'responseText: Done! A large White Chocolate Mocha with Almond milk and Cinnamon has been added to your order. Do you want to order Anything else?'* message to DialogFlow, after which DialogFlow responds to user with the payload 'responseText' received.
- The user can choose to say *'Yes'/'No'*. Yes results in a fulfillment request to the Flask server with a the following parameters:
  - Intent: Order-Intent-Followup-Yes

  Reply of 'No' would send a fulfillment request with the following parameters:
  - Intent: Order-Intent-Followup-No
- The server then receives corresponding calls and directs to custom functions that handle the corresponding intents with payload of parameters and the responses are handled accordingly. For example, if Order-Intent-Followup-No is triggered, the order is considered complete and the complete order details are fetched from the Firestore and listed out to the user.

vi. Integration with Google Assistant

To make our conversational agent more reachable to users, we need to expose it to popular platforms that are used by people. One such platform we chose to integrate with is Google Assistant, that enables any voice-powered devices such as Google Home, mobile phones that support it. To invoke our application from Google Assistant there are essentially two following ways:

- **Implicit and Explicit Invocation**: One way to provide an entry point into our application is via calling it by name (e.g. Hey! Talk to OrderBuddy.) with the help of Explicit Invocation that would map this welcome utterance to the intent corresponding to it. The other way would be to directly perform some task (e.g Can I have a small coffee!) with the help of Implicit Invocation that maps to the intent corresponding to the task.

  To integrate Google Actions with the chat bot developed using Dialogflow, an intent with event Google_Action_Welcome needs to be defined. When the end-user triggers our application with invocation, the context of our chat bot is invoked using this intent and from then on-wards every utterance is with the chat bot until the end of conversation.

vi.1 Personalization

We understand that the important selling point of a direct-to-customer application like OrderBuddy is Personalization. In order to provide customized recommendations for a repeat customer or just to address a new customer by his name in his next interaction with our application, we need to know his identity via primarily using 'Account Creation'.

- **Account Creation**: Leveraging the authentication systems provided by third-party systems such as Google is less tedious than building our own authentication system for account creation. Hence, we decided to use Google for this task, where the conversation control momentarily transitions into Google's environment to extract basic user information such as name or email, achieved using Helper intents from Action on Google developer platform .
- **Helper Intents**: Helper Intents allow Assistant to present a standard User Interface to users to obtain basic information. Created a Dialogflow intent that specifies the event corresponding to one of the helper intents. When we call a helper function, the Assistant presents the dialog corresponding to that helper. Upon receiving the user's response, the assistant returns the results of helper, and the intent corresponding to that event handles it.

  Google Actions leave the option to the application developer to when to invoke sign in helper intent. We analyzed the pros and cons of prompting for sign-in information at the beginning of conversation. Asking for a prompt at the beginning might drive away lot of new users as they are asked for personal details even without experiencing the feel of the application. Hence, we decided to ask for details only after the user places his first order. After the $actions_{INTENT\_SIGN\_IN}$ (Figure 6) event is triggered, the Assistant prompts the user for authorization. In case the user responds "Yes", the Assistant handles the account creation and prompts the user with account confirmation, else the guest-flow is provided.

vii. Testing Framework

In order to evaluate[7] how well the the voice based ordering system is performing, two main functionalities - intent detection and entity extraction were tested. To achieve this objective,

- A set of testing phrases consisting of all possible intents and entities had to be formulated along with the golden truth. In order to avoid bias in testing phrases, following were the different techniques used in collection of intent phrases:

1. Manual collection from a coffee shop: Since we are building a voice based chat bot for Starbucks, few of the team members spent time in the Starbucks cafe present in the campus and manually noted down the conversation a user has with the employee at the counter.

2. Friends and Family: Every person one or other time would have had an experience

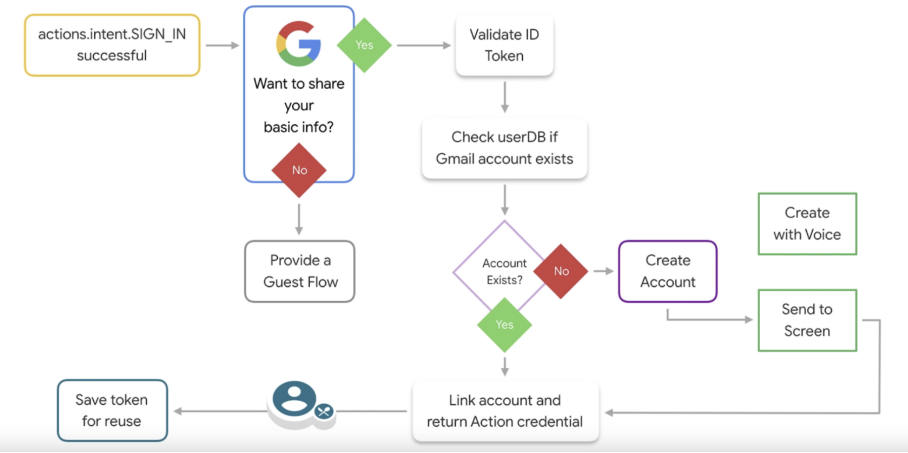

Figure 4: *Performing SIGN_IN Action*

of visiting the coffee shop and placing the order. Why not ask the people who will be a potential user of this application to help us with information of how they place, customize or cancel an item while placing the order. We sent a google form[3] and asked people to fill out their responses.

3. Taskmaster: It's a dataset[4] provided by Google consisting of 13,215 task-based dialogues in English, including 5,507 spoken and 7,708 written dialogues created with two distinct procedures. Each conversation falls into one of six domains: ordering pizza, creating auto repair appointments, setting up ride service, ordering movie tickets, ordering coffee drinks and making restaurant reservations. Conversation corresponding to ordering drinks were separated and used as a potential testing phrase.

Few of the examples collected from the above techniques include:

- "I'd like to order a small latte with some chocolate" - order intent with customization
- "Could I have Venti coffee, please?" - order intent

Since entity testing was also part of the idea, we replaced the drink, size customization actual values in the testing phrases with drink, size and customization placeholder as shown below so that we can formulate new testing phrases with all different possible combinations of drink, size and customize entity values.

- "I'd like to order a {size} {drink} with some {customize}"
- "Could I have {size} {drink}, please?"

Now that we have huge number of testing phrases, it is impossible to test manually for all of them. An automated testing framework which acts a client for the chat bot framework was developed in python using Pytest which takes in testing phrases formulated above and validates with the golden truth output.

Testing results can have two kind of errors broadly: 1. Intent Error: The detected intent is not same as the golden truth intent 2. Entity Error: Intent detection is right but entity detection - drink, size, customize parameter is wrong.

III. Results

A sample conversation in **Figure 5** shows the transition of intents based on user's utterances and context mapping, along with responses. As seen in the figure mentioned, a typical user wishes to have a small coffee in first figure of the flow. Upon receiving user's voice input, it

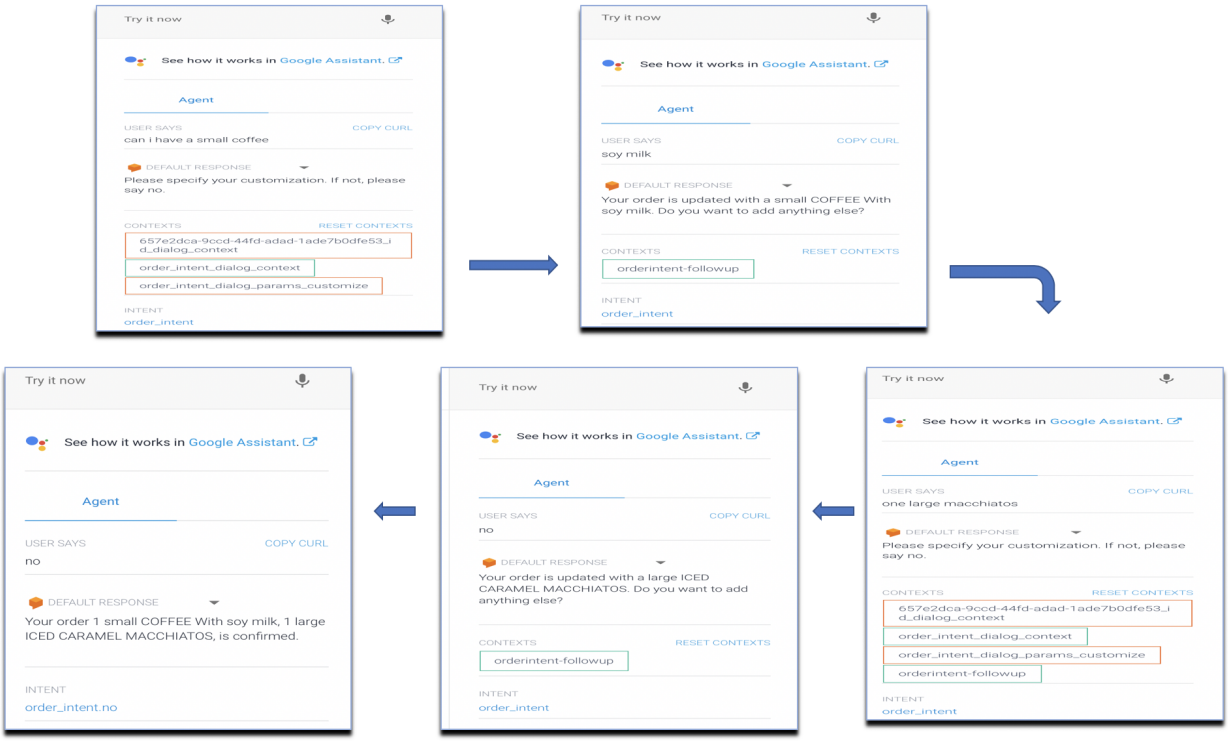

Figure 5: *A Sample Conversation using OrderBuddy*

is converted to text. The intent is identified and agent looks for required entities to fulfill the order and hence prompts the user to specify, i.e., customization (second figure in the flow). Once the slot-filling (required entities) is accomplished, the Firestore is updated with the current order information for the user as part of the business logic and the user is notified about the same, including in response if he wants to add anything more. The user adds another drink to the order (third figure in the flow), without any customization (fourth figure in the flow). Again the user is asked if they would like to add to the order. Here, the user does not want to add anything else to their order, which results in completion of the entire order of which the user is notified (fifth figure in the flow).

Each intent is evaluated separately and the number for training and testing phrases used for evaluation of each intent along with the accuracy scores for the same are listed in Figure 6. Order intent without/with customization achieved accuracy scores of 88.54% and 87.89% respectively. These errors were mainly entity recognition errors with no intent recognition errors. Cancel intent received a slightly lower accuracy of 83.44%. This was again due to the entity recognition errors mainly. As seen in the same figure, Cancel order and Complete order intents have received very high accuracy of 96% and 100% respectively. The reason is that we have reserved phrases for each of these intents to be recognized which are 'Abort' for Cancel order and 'Complete order' for Complete order intent to be recognized. There are no entities required to be identified for these intents, resulting in such high accuracy.

IV. Discussion

The results obtained show that we have managed to develop an application to provide the basic functionalities for a speech to text order

INTENT	TRAINING	TESTING	ERRORS	ACCURACY
Order Intent without Customization	120	23507	2693	88.54%
Order Intent with Customization	85	9772	1183	87.89%
Cancel Intent	50	2880	477	83.44%
Cancel Order	15	50	2	96%
Complete Order	15	50	0	100%

Figure 6: *Evaluation scores for each intent*

assistant. To make it more robust, expansion of our training set would be required to gain a better accuracy for the order and cancel item intents. Taking the results into consideration, as part of future work, the following tasks could be implemented:

- **Increase training for intents** We plan to expand our training set to make our model more robust.
- **Modification of existing order** We want to enable modification to orders, that provides ability to the user to be able to change already ordered items anytime during the order life-cycle.
- **Multiple Items and Restaurants** The plan is to expand this application to support multiple restaurants by including the menus from different restaurants, along with implementing a provision of accepting user's utterance with list of items required instead of just a single item at a time.
- **Personalization** As of now, a history of all past orders are stored and maintained in our database for every user. It would be helpful to use this data and provide a personalized experience to the customer on their next visit to the application.

V. Statement of Contributions

- **Aanchal Anil Samadariya** Design of Intents, Google Cloud Firestore, Fulfillment, Test framework and Dialogflow.
- **Apoorva Kasoju** Design of Intents, Google Cloud Firestore, Google Assitant Integration and Initial Research.
- **Karan Gulur Mularidhar** Design of Intents, Dialogflow, Google Assitant Integration, Test framework and Initial Research.
- **Rohith Krishna Sai Kanuparti** Design of Intents, Google Cloud Firestore, Test framework, Dialogflow and Fulfillment.

References

[1] Google Cloud Speech to Text: `https://cloud.google.com/speech-to-text/`.

[2] Google Cloud Natural Language: `https://cloud.google.com/natural-language/`.

[3] Google form: Link to Form

[4] Taskmaster dataset: `https://ai.google/tools/datasets/taskmaster-1`

[5] Google Cloud Dialogflow: `https://cloud.google.com/Dialogflow/docs/reference/rest/v2-overview`.

[6] Google Cloud Firestore: https://firebase.google.com/docs/Firestore/quickstart.

[7] Integration with Dialogflow: https://cloud.google.com/dialogflow/docs/integrations/.

[8] Brenes, David Gayo-Avello, Daniel and Pérez-González, Kilian *Survey and evaluation of query intent detection methods* Proceedings of Workshop on Web Search Click Data, WSCD'09. 10.1145/1507509.1507510

Appendix

Agent A natural language understanding module that handles conversations with end-users. It serves as top-level container for data and settings, including intents, entities, integration and fulfillment. Dialogflow provides pre-built agent which is specific to use-cases such as booking reservations.

Entity Intent parameter with type and possible values to extract data from an end-user's expression, including system entities that match date, time, color.

Intent Intent categorizes an end-user's intention. Intents consists of training phase, actions, parameters, response, contexts and events. When a user's utterance is matched to one of training examples, corresponding intent is matched along with extracted parameters and triggers associated action/text response.

- **Follow-up Intent** A follow-up intent is a child of its associated parent intent and contexts get automatically configured for this pair. When a follow-up intent is created, its input context and output context for parent intent is set with same name. A follow-up intent is only matched when the parent intent is matched in the previous conversational turn.

Context Context helps to understand user's utterance by controlling the order of intent matching based on input-output context mapping.

- **Input Context** Input context helps to match intent only if the end-user expression is a close match and if the context is active.
- **Output Context** Output context helps to activate a context if it's not already active or to maintain the context after the intent is matched.

Fulfillment Allows real-time, dynamic responses and triggers action for user's utterances by connecting your service to the agent for integration. To achieve this, intent should have fulfillment enabled else intent's default response is returned.

i. Intent Definitions

- **Order Intent:** This intent handles the ordering intent as seen in **Figure 6 (*left*)**. Adds an item to the current outstanding order of the user.
  - **Sample Utterances:** Can I have a large coffee?; please add a flat white
  - **Input Context:** None expected.
  - **Output Context:** *order-intent-followup*
  - **Context lifetime:** 1 conversation
  - **Response:** Done! do you need anything else?
  - **Fulfillment:** Add the item to Firestore collection
- **Order Intent followup - Yes:** This is a follow up intent which follows after the order intent, if the user replies *Yes* to the question *Do you want to add anything else to the order* then the Dialogflow agent resolves to this intent.
  - **Sample Utterances:** yes ; yes, a large coffee;
  - **Input Context:** *order-intent-followup*
  - **Output Context:** None
  - **Context lifetime:** NA
  - **Response:** what do you want to add?
  - **Fulfillment:** None
- **Order Intent followup - No:** As seen in **Figure 6 (*right*)**,this is a follow up intent which follows after the order intent, if the user replies *No* to the question *Do you want to add anything else to the order* then the Dialogflow agent resolves to this intent.
  - **Sample Utterances:** No; No I do not want to add anything;
  - **Input Context:** *order-intent-followup*
  - **Output Context:** None
  - **Context lifetime:** NA
  - **Response:** Done your *ORDER* is confirmed
  - **Fulfillment:** Confirm and Delete the order from *current order* collections and added to *history* collection.
- **Complete Order Intent:** This intent is triggered when the user voices a utterance that conveys an intent to place the order

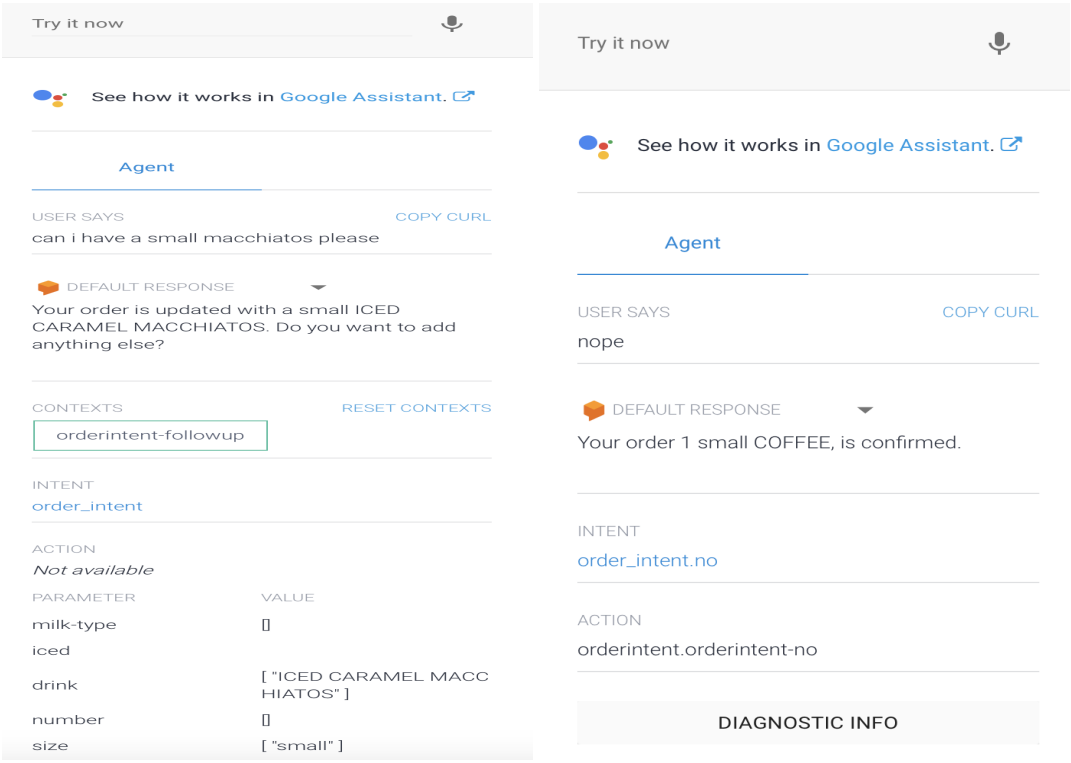

Figure 7: *On left:Order Intent; On right:Order Intent Followup - No*

as shown in **Figure 7 (*left*)** and its follow-up in **Figure 7 (*right*)**.

  - **Sample Utterances:** confirm order; complete order;
  - **Input Context:** None
  - **Output Context:** *complete-order-followup*
  - **Context lifetime:** 1 conversation
  - **Response:** Are you sure?
  - **Fulfillment:** None.

- **Cancel Order Intent:** This intent is triggered when the user voices a utterance that conveys an intent to cancel the order as illustrated in **Figure 8 (*left*)**.

  - **Sample Utterances:** abort; exit; bye
  - **Input Context:** None
  - **Output Context:** *cancel-order-followup*
  - **Context lifetime:** 1 conversation
  - **Response:** Are you sure?
  - **Fulfillment:** None.

- **Cancel Order Intent Followup - Yes:** This is a follow up intent which follows after the cancel order intent. If the user replies *Yes* to the question *Are you sure you want to cancel the order?*, the Dialogflow agent resolves to this intent.

  - **Sample Utterances:** Yes; yup;
  - **Input Context:** *cancel-order-followup*
  - **Output Context:** None.
  - **Context lifetime:** NA
  - **Response:** Done your order has been cancelled.
  - **Fulfillment:** Order is deleted from Firestore.

- **Cancel Order Intent Followup - No:** This is a follow up intent which follows after the cancel order intent. If the user replies *No* to the question *Are you sure you want to cancel the order?*, the Dialogflow agent resolves to this intent.

  - **Sample Utterances:** No; Do Not;
  - **Input Context:** *cancel-order-followup*

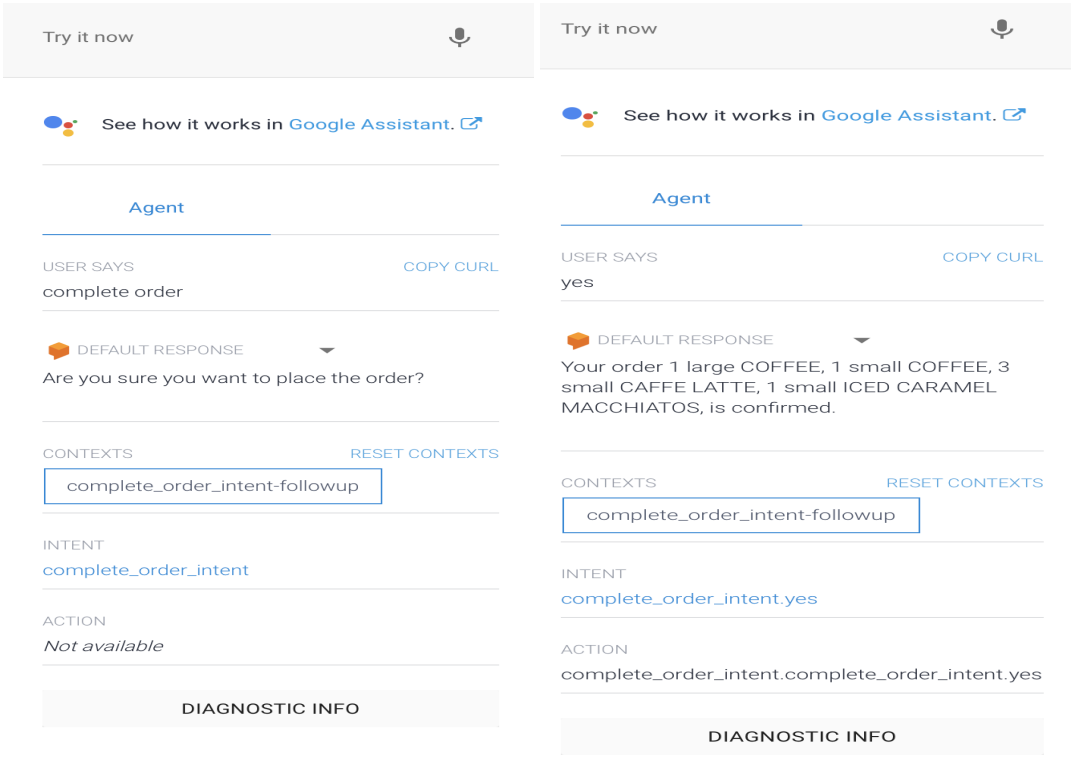

Figure 8: *On left:Complete Order Intent; On right:Complete Order Intent Followup - Yes*

  - **Output Context:** None.
  - **Context lifetime:** NA
  - **Response:** Do you want to add anything else?
  - **Fulfillment:** None

- **Cancel Item Intent:** When issuing a cancel item, user through his utterance can choose to issue the cancel intent in the following ways:

  - **With no drink parameter in the utterance** *Ex: Cancel; remove*
    - **Input Context:** None
    - **Output Context:** *Cancel-item-followup.*
    - **Context lifetime:** 1 conversation
    - **Response:** *Say 1 to delete your last item*
    - **Fulfillment:** Retrieve the drinks to be used for deletion.
  - **With a invalid drink parameter**, i.e the drink entity is recognized but has not been ordered by the user yet. *Ex: Cancel flat white; remove coffee*
    - **Input Context:** None
    - **Output Context:** *Cancel-item-followup.*
    - **Context lifetime:** 1 conversation
    - **Response:** *This drink is not in your order please say cancel followed a drink in your order*
    - **Fulfillment:** Retrieve the drinks present in the order to be used for response.
  - **With a valid drink parameter** *Ex: Cancel flat white; remove coffee*
    - **Input Context:** None
    - **Output Context:** *Cancel-item-followup.*
    - **Context lifetime:** 1 conversation
    - **Response:** Lists all the drinks in the given category to be deleted by the user.
    - **Fulfillment:** Retrieve the drinks

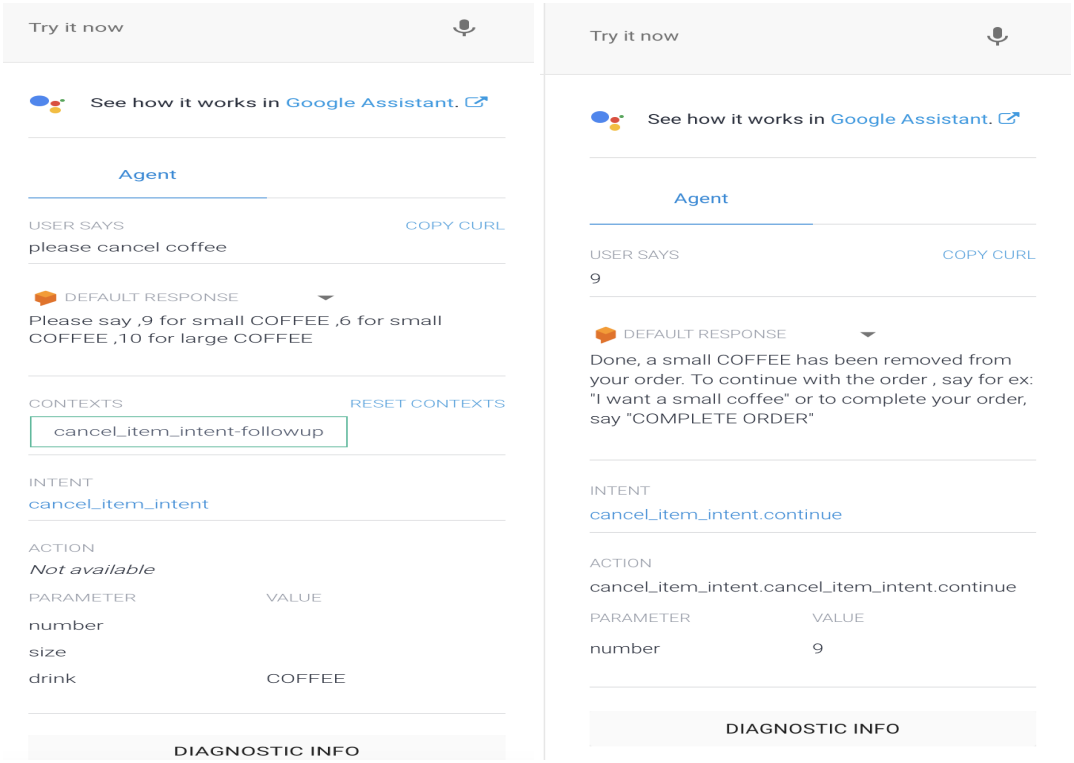

Figure 9: *On left:Cancel Item Intent; On right:Cancel Item Continue Intent*

present in the order to be used for response.

- **Cancel Item Intent Followup - Continue:** As seen in **Figure 8 (*right*)** ,this is a follow up intent which follows after the cancel order intent. If the user replies *Item-number* to the question *Please say 1 for large coffee, 3 for small flat white*, the Dialogflow agent resolves to this intent.
  - **Sample Utterances:** 13; item number 3;
  - **Input Context:** *cancel-item-followup*
  - **Output Context:** None.
  - **Context lifetime:** NA
  - **Response:** If the item is present it is deleted or we repeat the drinks list to the user
  - **Fulfillment:** Item is deleted if present in the order or user is prompted to choose from the list of drinks again.
- **Cancel Item Intent Followup - No:** This is a follow up intent which follows after the cancel order intent. If the user replies *No* to the question *You sure you want to delete an item? Please say 1 for large coffee, 3 for small flat white*, the Dialogflow agent resolves to this intent.
  - **Sample Utterances:** No;
  - **Input Context:** *cancel-item-followup*
  - **Output Context:** None.
  - **Context lifetime:** NA
  - **Response:** We repeat the drinks list to the user.
  - **Fulfillment:**Retrieve the drinks ordered by the user to choose.