Human-Machine Dialogue Project Report

Jacopo Donà (229369)

University of Trento

jacopo.dona@studenti.unitn.it

1. Introduction

This report delves into the implementation and utilization of the Rasa framework for developing a chatbot-driven pizza ordering system. Rasa, a powerful open-source platform for building AI-driven conversational agents, provides a versatile framework that enables developers to create personalized and engaging interactions with their users. Leveraging Rasa's capabilities, this project aims to design and construct a pizza ordering system that offers a seamless and interactive user experience, showcasing the potential of AI-driven solutions in enhancing the food ordering industry. By leveraging these tools, small companies can manage multiple ordering requests at the same time, providing them a scalability opportunity that would otherwise not be available if only a human operator was in charge of registering the orders from clients.

For the target-group of the developed agent, I tried to make sure the bot is as flexible as possible in assisting a user, meaning the goal is to guide less experienced users in completing an order with a more supportive approach, while letting users that are already familiar with automated food ordering compose requests that are filled with multiple details at once, in order to speed up the ordering process.

The proposed conversational agent is able to handle multiple tasks that are both composed of question-answer type of interaction and multiple turn-taking dialogues in the realm of pizza ordering. In particular, I grouped the available actions the bot can perform into 3 main areas:

1.1. Task 1: Pizza and Drink Ordering

The fundamental tasks of this system is to seamlessly facilitate the process of selecting and customizing pizza and drink orders. The system offers dynamic interactions able to accommodate users that are both inexperienced (with under-informative requests tendencies and little item knowledge) and users that are already familiar with the tool (higher domain knowledge and requests richer in information).

For the pizzas, entities like the pizza name and the size are asked while making an order, and the user also has the possibility of adding an extra topping to their desired pizza while formulating their request. For the drinks, the option of ordering multiple bottles of beverage at the same time is provided.

Once the order has been placed with either one or multiple items, the system guides the user in the checkout process, which supports both takeaway and delivery options. Based on the user's preferred method, different entities are required to complete the registration and, once they are correctly given, the order is saved on the server.

1.2. Task 2: Information Inquiry

In this task I group a series of questions and inquiries the user could ask to the bot in order to gather information about the restaurant and the available items.

The user can in fact inquiry the system on a series of information like the pizza menu, their sizes, their price and which ingredients are included in a pizza. In addition, the user can also asks for pizza suggestions, in particular by asking the system for pizzas that contain or do not contain a particular ingredient.

Finally, the user can also inquiry for the list of available drinks, their price, and ask checkout information such as the opening hours or the location of the pizzeria. An help request is also available, that aims to help the user based on the status of his conversation.

1.3. Task 3: Order management

In this group I report all the requests a user can ask in order to modify a previously set order.

In particular, the user can easily request to add or remove pizza/drinks from his previously set order, modify information about the checkout such as the name of the delivery door bell or of the person who will pick up the order, time of the pickup/delivery or change the checkout method altogether. Both during the ordering phase or after the order was placed, the system can remind the user what is order is currently consisted of by recapping the ordered items, the total price and, if the checkout method was already expressed, also the details on time of delivery, name and address of the order. The user has also the possibility of deleting his previously set order if he wishes, however in a real scenario some internal controls should be integrated to make sure this option is no more available if the order is being prepared by the pizzeria.

2. Conversation Design

2.1. Initiatives and Ground Taking

As seen during the lectures, designing how the initiative is managed is critical for the interaction design:

- a system-initiative conversation will be composed of almost all question-answer type of interactions, feeling way too robotic and possibly restricting the user freedom in deciding what to do.
- a user-initiative requires the user to know exactly what he wants to do, as the system will provide little guidance in the conversation, thus tends to be confusing for users with less experience.

For these reasons, I decided to opt for a mixed initiative dialogue flow.

In this way, more experienced users have the freedom to guide most of the conversation, inserting items without too many repetitive question-answer interactions. Inexperienced users can also interact with the bot, and if the requests are identified to be without required entities, the system takes control of the conversation flow, asking and guiding the user in providing him his preferences. For further assisting newer users, I made sure to develop help utterances that variate based on the conversation status (example: if an order is yet to be made, if the user has started inserting items or if the user has completed his order with all the required information), in order to be able to give dynamic support to users throughout the conversation based on their current needs. At any point in the conversation the user can ask the bot for assistance.

2.2. Acknowledgement

I firstly designed the bot for having implicit confirmation of each entity for the tasks of ordering food and checking out the order. In my opinion, this solution works best for chat-bots who work through textual chats only, and since in the early stages I focused on using the command line to test my bot, it was good for making sure the system was understanding the requests correctly. However, once I first deployed the bot onto the Alexa Connector, I noticed having an implicit confirmation every time an entity was provided was making the whole interaction take a lot of time, with the bot talking for the vast majority of the conversation. Since I believe it to be counterproductive, I removed some of the implicit confirmation when the user was entering part of the order, and use an explicit confirmation once all the slots of a form were filled to make sure the bot had understood correctly. In this way experienced user don't have to listen their inserted entities repetitively, and still use a single explicit confirmation to make sure the entities were correctly gathered by the bot.

2.3. Error Recovery and Fall-back Responses

As mentioned in the Acknowledgement section, I decided to reduce the number of implicit confirmation of inserted entities. To make sure the user and the bot understand the conversation parameters in the same way, I inserted explicit confirmations once a request or modification form is complete. If the users realizes the bot misunderstood something, it can explicitly tell the bot, who restarts the request and clears the previously inserted values.

Furthermore, a user can stop any form he has started by saying to the bot he has changed his mind, thus allowing the user to follow "unhappy paths" while filling a form, avoiding to have the user stuck in a form-submitting loop.

In addition, through global slots and custom actions I implemented several controls to make sure the conversation flows normally. When the users asks for an pizza/drink, a check with the knowledge base is made to make sure the request item actually exists, and if it does not the system alerts the user (in addition, in case an item does not exist, the system implicitly suggests the user to check for the available items in the menu).

Some controls are also made to make sure the conversation flow is correct, for example, a user cannot check out or delete an order he has not made yet, and in that case the system blocks the user before a request is carried out.

I encountered some difficulties in dealing with out-of-scope requests, mainly because it is difficult for the bot to realize when the user is asking something it cannot do, since it may contain terms never seen during training. I added an out-of-scope intent in the nlu file and used a rule to deny user requests that fall in its category. In addition, I also integrated a Fallback Policy, which intervenes in case the classified intent has a low score or is ambiguous with another intent within a certain threshold. This policy consists in a one stage fallback, which asks the user to rephrase its last request. During testing, I noticed that most of the time unexpected sentences trigger the Fallback policy rather than the out of scope intent.

With respect to server errors, the system informs the user something has gone wrong if some error occurs internally. Situations like the server not being up is not treated as of now due to the server being run on a local-host port, but if the server was to be deployed on an external machine, it would require a check at each interaction to make sure the server is online and receiving requests correctly.

2.4. Custom Actions

The bot makes use of a large number of custom actions implemented in the `actions.py` file, which are used to link user requests to the server where interactions with the knowledge base are executed. In the actions, some controls on the previous status of the conversation are made, in particular the *ResponsePositive* and *ResponseNegative* actions use some checks on the previous bot action to understand what the user is referring to, and some actions check global slots and order information to make sure the user requests have an effect if they are placed in the correct stage of the conversation.

The global slots consists of boolean variables, which are used to insert in the tracker slot information on whether the user has an order with items and if he has completed an order. They start by default at False and their value can be modified by actions which add/delete items.

As of now, I use the tracker attribute `sender_id` to register the orders, making sure the user can only interact with his order, thus allowing the system to interact with multiple users at once. This solution works fine while the bot action server stays online, but due to the id of a user device possibly changing through system reboots (for example, by shutting down and restarting a `rasa shell`), a login strategy could be more effective in keeping track of the user for longer periods of time and should be considered as future upgrade.

3. Data Description & Analysis

3.1. Dialogue Data

The dialogue data used was generated manually by imitating real conversations, annotating and converting them into a rasa-compatible format. The data is composed of 34 stories and 42 rules.

I used rules to handle requests that involve a single question-answer interaction and for form initialization and submission. Stories, on the other hand, are used to teach the system the system the typical conversational flow, adding also some unhappy paths to deal with unexpected changes in the user requests.

The number of turns of each dialogue varies on how much information is given by the user at each interaction:

- Requests for ordering and removing pizza or drinks can variate between 2 and 4, based on the level of details given by user on his desired entities.
- The dialogues for inserting pickup information last 4 turns, while the ones for delivery take 6 turns. Again, the user can make the interaction last less turns if he provides multiple entities in a single request.
- Interactions for changing order details take 2 turns.
- Requests to delete an order take 2 turns.

The total dialogue length is therefore variable and based on the user type and the size of the desired order.

3.2. NLU Data

NLU data comprises the training information used to assist the NLU model in comprehending and extracting organized details from user inputs. Generally, NLU data is composed of two primary elements: intents and slots. Intent classification and slot extraction are thus the two fundamental tasks in a NLU pipeline, and they can be performed jointly or independently from one another.

The data was mostly gathered manually, either by writing them explicitly or by using some of the user sentences gathered through the human evaluation. Some on the samples were also taken from the available baseline provided. In order to generate sentences to train the entity extractor, in particular for ingredients and timings, I re-used sentences template and changed the entity value by using a custom string manipulation script, synthetically generating data sentences allowing the model to learn what the entities typically consisted of. To aid this process, I also declared lookup tables and synonyms the model uses to match extracted entities to known ones.

The dataset is composed of 912 samples, categorized in 36 intents and 9 types of entities, which are listed in the Appendix.

3.3. Domain Data

As for the knowledge base, I decided to create a small database consisting of pizza and drinks manually made.

I use a python file called `classes.py` to define a series of object classes to use to contain information about orders, standard pizza and drink information, and for the latter two I also created a *”Ordered”* class variant to store details about a ordered item such as the pizza size, the amount of a specific item requested by the order (if the user orders more than one) or the customization (if the user decides to add an extra topping to a pizza).

The menu is composed by 12 pizzas and 5 drinks. Each pizza contains from 2 to 5 ingredients, and it can be ordered in small, medium or large size. The action server automatically computes the order price and each modification made to a pizza such as toppings and size modifies the price of the item.

For the checkout, I assume the pizzeria to be located in Trento, so the bot expects to receive an address in Italian format, some examples are ”Via Rosmini”, ”Piazza Duomo”, ”Viale Verona” or ”Corso Alpini”. The bot is able to understand address not explicitly inserted in the nlu training file, however due to time and complexity constraints I did not integrate a check to make sure the address actually exists and is placed in the urban area of Trento.

4. Conversation Model

The conversational agent has been developed with Rasa, in the `config.yml` files I provided multiple configurations for both the NLU and DM pipeline.

4.1. NLU model

As mentioned before, the slot filling and intent classifcation can be performed jointly or separately. To test both scenarios, I developed 2 different configurations files. One uses the rasa Dual Intent Entity Transformer (DIET) [3] to jointly perform entity extraction and intent classification, while the other one uses the SKLearnIntentClassifier for performing intent classification and a CRFEntityExtractor for performing entity extraction.

Both of these configuration use a pretrained language model, spaCy’s `en_core_web_md` [2]. This choice was made because I believe the language domain of the typical pizza ordering chatbot is not too specific to require an ad-hoc trained language model, which would also require a much larger training dataset. The Spacy model is used to perform tokenization and featurization. The tokenizer allows for the analyzed text to be broken down into words and subwords, named ”tokens”. The Spacy featurizer uses the obtained text tokens to generate dense feature representation for the analyzed utterances, allowing the model to capture semantic relationships between words. This featurizer is also paired with the RegexFeaturizer, LexicalSyntacticFeaturizer and CountVectorsFeaturizer, which instead are sparse featurizers.

4.2. Policy Configuration

The Policy configuration is defined as such:

- RulePolicy: this policy predicts the best action by checking the rules defined in the `rules.yml` file along with their conditions
- MemoizationPolicy: this policy predicts the best action by trying to match the current conversation with the learned stories defined in the `stories.yml` file
- TEDPolicy: this policy uses a machine learning approach to allow the bot to generalize to unseen user inputs.

In a separate configuration file, I also tried replacing the MemoizationPolicy with the ”Augmented” variant, which employs a forgetting mechanism that will forget up to a certain amount of steps in the conversation history and try to find a match in the training stories. The policies act in parallel, with each policy making a prediction on the action to take at each step of the conversation, and the action with the highest confidence is executed. In case policies predict different actions with the same confidence, then a priority order is followed. In the configuration, the policy with the highest priority is the RulePolicy, followed by the MemoizationPolicy and at last the TEDPolicy. As suggested by the Rasa developers, I did not modify the policy priority.

4.3. Deployment

Once all the tasks were completed, I deployed the bot on Alexa the using the Amazon Developer Console. This allowed to integrate the Dialogue Management, NLU and Dialogue Generation tasks done by Rasa with the Automatic Speech Recognition and Text to Speech tasks, which are independently carried out by Amazon Web Services. I was not able to test the bot on an actual Amazon Echo speaker, but I used the Amazon Developer Console test section where it is possible to interact with the deployed Amazon skill using the keyboard, with Alexa performing the Text to Speech for the bot responses.

5. Evaluation

5.1. NLU Evaluation

To evaluate the NLU pipelines, I decided to use a hold-out strategy, where I excluded 20% of the training data and used it as test set, while the pipeline was trained on the remaining 80%. Due to the small dataset sized, I decided to run this procedure 5 times and report the mean and standard deviation of the results. The metrics I choose were the accuracy for the Intent Classification, and the micro-average F1 score for the Slot-Filling. I chose to use the micro-average due to the entities having different distributions in the test set, and I wanted to achieve a fair comparison where each sample contributed equally to the final score. A comparison for both pipelines is available in Table 1. The joint architecture of the first configuration file is better at Intent classification, but is slightly worse in extracting entities w.r.t the CRF.

Testing the conversational agent produced by the two pipelines, I noticed that the configuration using the SKLearn classifier was producing correct intent classifications, but the confidence levels were quite low, which caused the bot to use the fallback action more often. This behavior is also evident during the testing of the NLU pipeline, as shown in Figures 2, where we can see the predictions are correct but the confidence levels are low. The distribution of the DIET classifier is available at Figure 1 and exhibits a more confident behavior.

This is somewhat to be expected, as we are comparing a DIETClassifier which is a multi-task transformer with the SKLearnIntentClassifier, which instead uses a Support Vector Machine for the classification; the former benefits from the inherent contextual understanding and representation learning capabilities of transformers, whereas the latter relies on traditional feature engineering and exhibits a lower capacity in capturing complex linguistic nuances.

As a small note, while testing the bot on the Alexa Connector I noticed a small degrade in the bot entity recognition capabilities. This is due to the fact that the Alexa platform, even when using the keyboard input chat, modifies part of the inserted text. In particular, I noticed capital letters on names, time formats and certain words are post-processed by the Alexa platform and missed by the rasa NLU pipeline (example: slot "Jacopo Donà" becomes "jacopo dona", "20.30" becomes "20 point 30" and "Number 30" becomes "#30"). I tried enriching the nlu training data with synonyms and further formats, but in particular the full names are sometimes not extracted correctly, with only part of the name being identified. This causes the agent to ask again the user's full name, as it expects values composed of first and last name. These problems do not show up when using rasa shell, where the input text is sent directly to the NLU pipeline without any pre-processing.

5.2. Dialogue Management Evaluation

To evaluate the Dialogue Manager, I created a set of 25 test stories. These stories involve user input, and are constructed to emulate both stories that are used to train the agent and stories which have some combination of dialogues not seen by the Policies during training, with the goal of challenging the system to handle known but also potentially new user paths. The evaluation results were initially mixed, as the MemoizationPolicy was able to solve correctly all test cases, while the Augmented version miss-handled two stories containing the `init_checkout_request` intent. I believe this issue was caused by the fact that the actions in response to this intent were only declared through stories, and the Augmented Memoization could not find a corresponding match in the training stories (which did not cover all the possible combinations that can lead to the checkout request) due to the history parameter. To solve this, I added the `init_checkout_request` to the rules, paired with a condition check to the slot that tracks the conversation status. This fix did not affect the perfect accuracy of the MemoizationPolicy, but it allowed the Augmented variant to correct its behavior, leading it to completing as well all the test stories successfully.

5.3. Human Evaluation

During the development of this project, I conducted 2 human evaluation testing phases: one during the mid-stages of the development of the bot where just TASK 1 and TASK 2 were integrated (note: some functionalities, for example the request of the street number in the delivery form, were added in a later stage), and a final one once all the tasks had been implemented.

The human evaluation was conducted both times by 3 users who are not Artificial Intelligence students, thus unfamiliar with any notion of Natural Language Understanding and Dialogue Management.

In the first phase I conducted the testing via command line. I briefly explained the context of the bot and the available actions, without giving any guidance on how a sentence should be formulated.

The bot was able to successfully solve the tasks, but some minor errors were encountered in the checkout phase and in the ordering phase. In particular, the slots for the hours were not robust in being given in different format (example: 8pm vs 20.00) so the bot would ask the user to repeat them. Another error that was found is that during the request of the delivery address, due to the fact some Italian street addresses are composed by "via [name] [surname]" sometimes the name of the street would be automatically registered as the name of the user, and the bot would just save the address as "via". Regards to the ordering phase, if the user was over-informative on the pizza request, sometimes the bot would not respond.

After solving the errors and completing the bot functionalities, I conducted a second test with the human evaluators, this time through the Alexa Developer Console.

The only indication I gave, due to the issues discussed in subsection **5.1**, was to give the time of the order in English format (example 8pm), but the users were independent in formulating their requests and interacting with the agent.

Again, the bot was able to successfully complete the tasks requested by the users. It is to note that the delivery form can incur in some error, in particular the street names and the client names are sometimes miss-interpreted, and I believe a training dataset with more examples could improve the performance. Also, adding an ad-hoc entity extractor focusing on Italian streets and names could be considered for a future improvement.

Another flaw that was noticed is that the bot, when tested through the command line, is not too robust to typing errors, in particular if multiple words contain some small typo, the agent triggers the Fallback Classifier due to low confidence scores. In the case of a Voice Assistant with ASR this is not a problem, but if the bot was to be deployed as a text-assistant, then some fine-tuning should be made.

6. Conclusion

In conclusion, in this project I developed and discussed the conversational agent applied to the domain of automated food ordering. By conducting tests on NLU and Dialogue Management, I provided a metric-guided evaluation of the system, and explained some of the errors. Through deployment and human evaluation I discovered how some minor errors are introduced in the pipeline by the Alexa Connector, which leave room for improvement for the agent in future updates. The code of this project is available on my Github repository [4], and the flowchart of a human-agent conversation is available in the appendix at Figure 3.

7. References

[1] T. Bocklisch, J. Faulkner, N. Pawlowski, A. Nichol, “Rasa: Open Source Language Understanding and Dialogue Management”, 2017. Documentation available here

[2] SpaCy, en_core_web_md documentation available here.

[3] T. Bunk, D. Varshneya, V. Vlasov, A. Nichol, “DIET: Lightweight Language Understanding for Dialogue Systems”, 2020.

[4] My code is available here.

A. Appendix A

Configuration	Intent Accuracy	Slot F1
Config	89.2 ± 0.2	91.1 ± 0.2
Config 2	85.9 ± 0.2	92.5 ± 0.1

Table 1: *NLU Pipeline comparison. Config uses DIET to jointly solve the tasks, while Config 2 uses SKLearnIntentClassifier and CRFEntityExtractor separately*

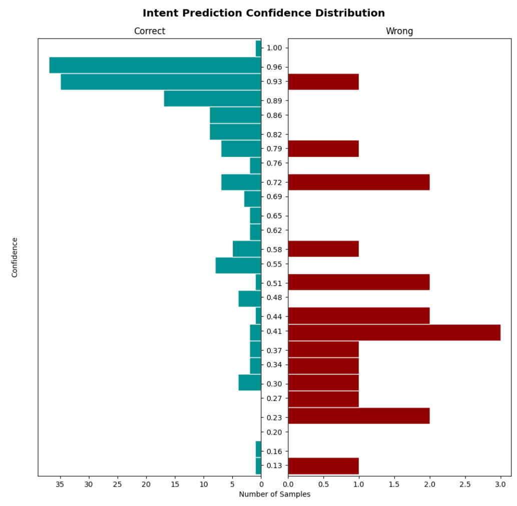


Figure 1: *DIET intent prediction confidence distribution*

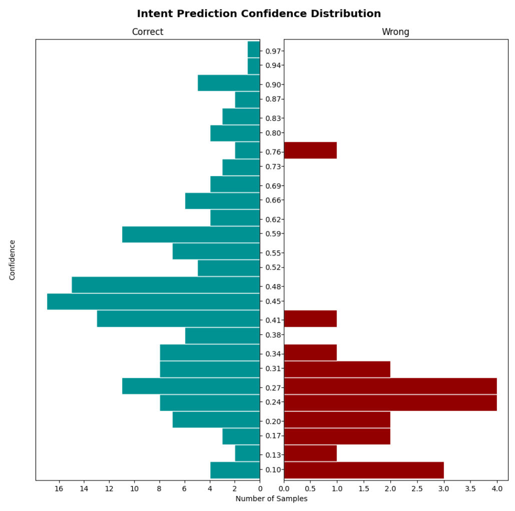


Figure 2: *SKLearnIntentClassifier prediction confidence distribution*

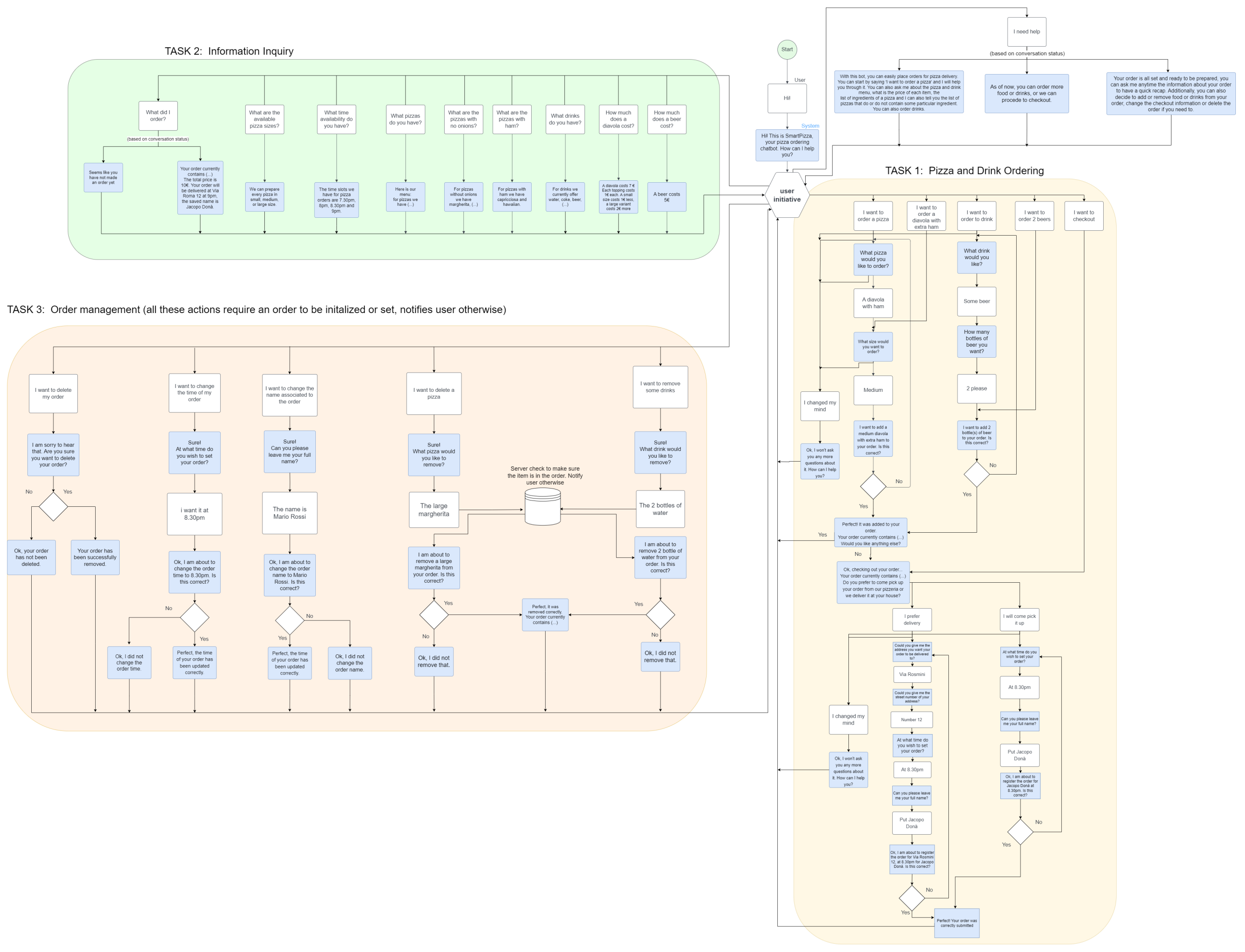
List of intents:

- welcome_greet
- goodbye
- help
- response_negative
- response_positive
- out_of_scope
- stop
- init_request
- init_drink_request
- ask_drink_list
- ask_drink_price
- say_pizza_type
- say_pizza_size
- say_drink_name
- say_drink_amount
- init_checkout_request
- inform_order_delivery
- inform_order_pickup
- say_order_time
- say_full_name
- say_home_address
- say_address_number
- ask_pizza_menu
- ask_pizza_price
- ask_pizza_ingredients
- ask_pizza_with_ingredient
- ask_pizza_without_ingredient
- ask_pizzeria_location
- ask_time_slots
- ask_pizza_sizes
- ask_order_info
- delete_order
- modify_order_time
- modify_order_name
- remove_pizza
- remove_drink

List of entities:

- pizza_size
- pizza_type
- drink_name
- drink_amount
- client_name
- order_time
- address_street
- address_number
- ingredient

Figure 3: *Flowchart of the conversational agent*



TASK 2: Information Inquiry
What did I order?
(based on conversation status)
Seems like you have not made an order yet
Your order currently contains (...) The total price is 10€. Your order will be delivered at Via Roma 12 at 9pm, the saved name is Jacopo Donà.
What are the available pizza sizes?
We can prepare every pizza in small, medium, or large size.
What time availability do you have?
The time slots we have for pizza orders are 7.30pm, 8pm, 8.30pm and 9pm.
What pizzas do you have?
Here is our menu: for pizzas we have (...)
What are the pizzas with no onions?
For pizzas without onions we have margherita, (...)
What are the pizzas with ham?
For pizzas with ham we have capricciosa and hawaiian.
What drinks do you have?
For drinks we currently offer water, coke, beer, (...)
How much does a diavola cost?
A diavola costs 7 € Each topping costs 1€ each. A small size costs 1€ less, a large variant costs 2€ more
How much does a beer cost?
A beer costs 5€
Start
User
Hi!
System
Hi! This is SmartPizza, your pizza ordering chatbot. How can I help you?
user initiative
I need help
(based on conversation status)
With this bot, you can easily place orders for pizza delivery. You can start by saying 'I want to order a pizza' and I will help you through it. You can also ask me about the pizza and drink menu, what is the price of each item, the list of ingredients of a pizza and I can also tell you the list of pizzas that do or do not contain some particular ingredient. You can also order drinks.
As of now, you can order more food or drinks, or we can procede to checkout.
Your order is all set and ready to be prepared, you can ask me anytime the information about your order to have a quick recap. Additionally, you can also decide to add or remove food or drinks from your order, change the checkout information or delete the order if you need to.
TASK 1: Pizza and Drink Ordering
I want to order a pizza
I want to order a diavola with extra ham
I want to order to drink
I want to order 2 beers
I want to checkout
What pizza would you like to order?
A diavola with ham
What size would you want to order?
Medium
I want to add a medium diavola with extra ham to your order. Is this correct?
What drink would you like?
Some beer
How many bottles of beer you want?
2 please
I want to add 2 bottle(s) of beer to your order. Is this correct?
I changed my mind
Ok, I won't ask you any more questions about it. How can I help you?
No
Yes
Perfect! It was added to your order. Your order currently contains (...) Would you like anything else?
Ok, checking out your order... Your order currently contains (...) Do you prefer to come pick up your order from our pizzeria or we deliver it at your house?
I prefer delivery
I will come pick it up
Could you give me the address you want your order to be delivered to?
Via Rosmini
Could you give me the street number of your address?
Number 12
At what time do you wish to set your order?
At 8.30pm
Can you please leave me your full name?
Put Jacopo Donà
Ok, I am about to register the order for Via Rosmini 12, at 8.30pm for Jacopo Donà. Is this correct?
Ok, I am about to register the order for Jacopo Donà at 8.30pm. Is this correct?
Perfect! Your order was correctly submitted
TASK 3: Order management (all these actions require an order to be initalized or set, notifies user otherwise)
I want to delete my order
I am sorry to hear that. Are you sure you want to delete your order?
Ok, your order has not been deleted.
Your order has been successfully removed.
I want to change the time of my order
Sure! At what time do you wish to set your order?
i want it at 8.30pm
Ok, I am about to change the order time to 8.30pm. Is this correct?
Ok, I did not change the order time.
Perfect, the time of your order has been updated correctly.
I want to change the name associated to the order
Sure! Can you please leave me your full name?
The name is Mario Rossi
Ok, I am about to change the order name to Mario Rossi. Is this correct?
Ok, I did not change the order name.
I want to delete a pizza
Sure! What pizza would you like to remove?
The large margherita
I am about to remove a large margherita from your order. Is this correct?
Ok, I did not remove that.
Server check to make sure the item is in the order. Notify user otherwise
Perfect, it was removed correctly. Your order currently contains (...)
I want to remove some drinks
Sure! What drink would you like to remove?
The 2 bottles of water
I am about to remove 2 bottle of water from your order. Is this correct?