Towards a Chatbot for Creating Trigger-Action Rules based on ChatGPT and Rasa

Simone Gallo [1,2], Alessio Malizia [2,3] and Fabio Paternò [1]

[1] *HIIS Laboratory, ISTI-CNR, Pisa, Italy*
[2] *Department of Computer Science, University of Pisa, Pisa, Italy*
[3] *Faculty of Logistics, Molde University College, Molde, Norway*

Abstract

In this paper, we propose a novel approach for building a conversational agent for creating trigger-action rules and controlling smart objects inside smart environments, such as a smart home. Our approach integrates ChatGPT, a state-of-the-art pre-trained language model for open-domain dialogue generation, with Rasa, a popular open-source framework for developing task-oriented chatbots. We leverage ChatGPT's abilities to perform Natural Language Processing tasks through prompting and few-shot learning, and Rasa Open Source's features to handle intents, entities, forms, and execute actions. We design Rasa custom actions that invoke ChatGPT's API to process complex customization rules, manage conversational breakdowns and answer questions about the smart environment.

Keywords

End-user Development, Conversational Agent, Generative Pre-Trained Model

1. Introduction

Recent advances in Natural Language Processing (NLP) have greatly improved conversational capabilities and language-related downstream tasks. Large Language Models (LLMs), such as GPT-3 and ChatGPT, have been instrumental in achieving those improvements. These models have enabled text generation, translation across various languages, text style rewriting, question answering, and more. With the use of prompt engineering techniques, they can perform zero-shot and few-shot tasks [1] from a small set of examples, without the need for training or fine-tuning a new model [2, 6].

Starting from previous work [4] in which we present a Dialogflow chatbot for creating customisation rules, we have worked on a new version based on Rasa in order to overcome the previous limitations (errors in interpreting complex rules, limited capacity to manage breakdowns), improve its capabilities and move to an open-source solution. Meanwhile, the expansion of LLMs and their skills in various types of tasks led us to consider a solution based on the combination of Rasa and ChatGPT. Large language models demonstrated advanced capabilities of understanding and generating natural language as well as maintaining context during the entire conversation, making it very close to a human and consequently more natural and less frustrating. However, applying these models to real-world problems requires further research since they exhibit probabilistic and not deterministic behaviour so even a small change in the input may lead to the generation of an unexpected or inaccurate output. A solution only based on ChatGPT – or more in general on LLMs – could raise well-known problems, such as "hallucinations" and bias, even for newer models such as GPT-4 (albeit a marked improvement over its predecessors) [8]. On the other hand, frameworks such as Rasa or Dialogflow are well-proven tools to build task-oriented chatbots and virtual assistants using machine learning to classify the user input into

IS-EUD 2023: 9th International Symposium on End-User Development, 6-8 June 2023, Cagliari, Italy
EMAIL: simone.gallo@isti.cnr.it (S. Gallo)
ORCID: 0000-0002-5162-0475 (S. Gallo)

[1] In **zero-shot** the model predicts the answer given only a natural language description of the task. In **one-shot** or **few-shot**, in addition to the task description, the model sees a single or multiple examples of the task.

an intent, extracting the relative entities. Then, using pre-defined rules or conversation patterns (e.g., Rasa stories) the system chooses the next action to be taken (e.g., execute code, making API calls), and generate or use predetermined answers for the user request. We propose a hybrid solution that can be the trade-off between exploiting the considerable capabilities of LLMs but at the same time mitigating their risks by integrating them with reliable tools for greater accuracy and control.

2. Related Work

Despite the most common commercial tools such as Amazon Alexa and Google Assistant do not support the creation of automation through natural language, various research solutions consider the conversational approach. For instance, Barricelli et al. [1] developed an Alexa skill that employs a multimodal approach (voice and touch) to guide users in selecting and refining triggers and actions for creating automation rules. Corno et al. [3] propose a chatbot that, based on the user's personalization intention, gathers user preferences during the conversation and ultimately suggests pre-existing IFTTT applets that closely align with the user's request. Lastly, Jarvis [7] is a Slack chatbot that enables the execution of instant or delayed commands on home devices in the form of trigger-action rules. Jarvis can compose rules with a maximum of one trigger and one action; thus, it is not possible, for example, to specify rules with actions that require a specific delay and, simultaneously, a certain condition to be verified in order to be triggered (e.g., "turn on the living room light at 7:00 pm if it is dark").

3. System Components Overview

This work proposes a tool for the creation of customization rules in trigger-action format by non-expert users for personalizing smart environment behaviour. Using natural language, the tool makes it possible to define events and/or conditions to be verified for the activation of the corresponding actions (e.g., "remind me to bring my umbrella when I leave home if it rains during the day"). Moreover, the proposed design can also be used to execute multiple instant actions in a single user command (e.g., "turn on relaxing light in the living room and play some music").

The Rasa component is used for the recognition of the user's intentions, the extraction of the relevant entities involved, and for managing the dialogue flow. ChatGPT components are employed for splitting complex rules (comprising multiple triggers and actions) into simple commands, managing breakdowns during rule creation, and providing answers and explanations consistent with the context of the smart environment in which the user operates (e.g., "what is a trigger?", "what are the available actions?", "what is the difference between an event and a condition?"). Moreover, ChatGPT can also be applied for answering explanation requests on why a certain action occurred at a certain time (e.g., "why did the heating system turn on?").

The tasks mentioned above can be realized by using prompting and one-shot or few-shot learning, without the need to collect and use datasets for training task-specific models. Furthermore, changes in the context (e.g., the addition of a trigger or an action in the intelligent environment) should be easily handled by modifying the prompt of the individual task.

The system architecture is shown in Fig. 1 and describes the workflow from a user automation request to the completed rule. When the chatbot receives an automation description, the message is sent to ChatGPT Splitter, returning the variables "tg" and "ac" that indicates the identified "triggers" and "actions" involved. The simpler commands obtained are then identified by the Rasa Intent Classifier, eventually starting the "slot-filling" phase (managed by the Rasa Dialogue Manager). A ChatGPT Classifier is used to manage possible breakdowns during the creation of the rule. Finally, a feedback message is given to the user when the rule is complete.

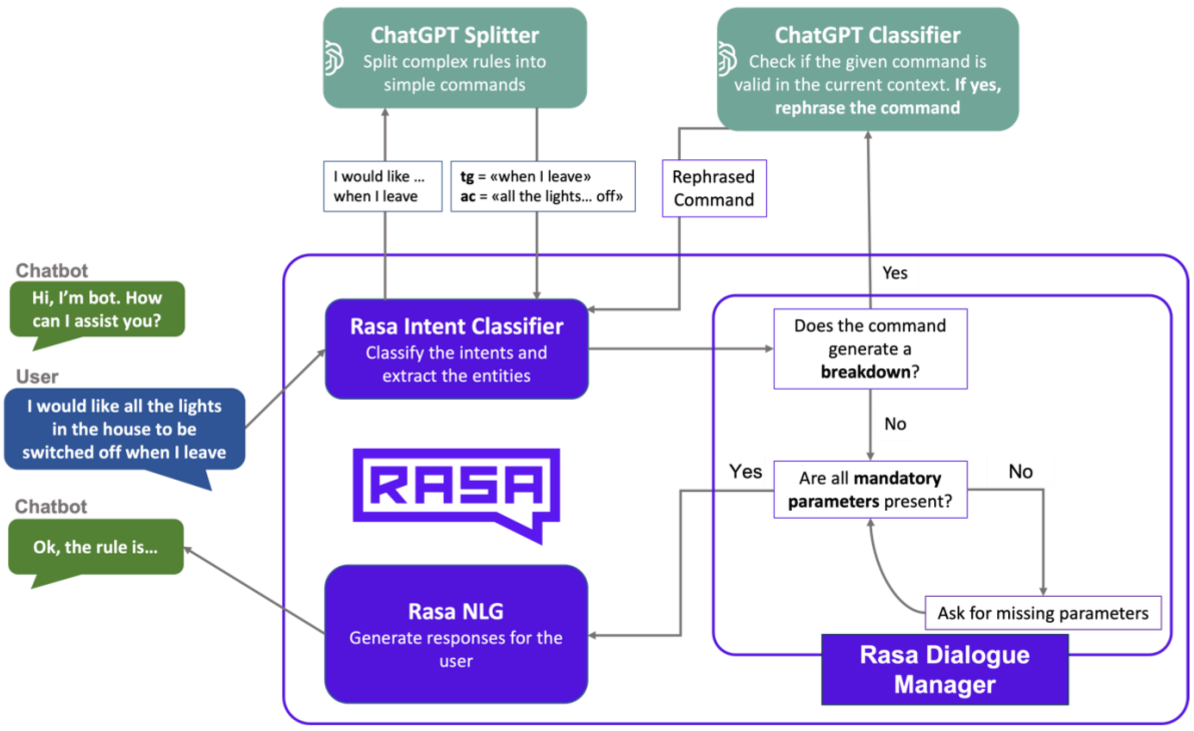
ChatGPT Splitter
Split complex rules into simple commands
ChatGPT Classifier
Check if the given command is valid in the current context. If yes, rephrase the command
I would like ... when I leave
tg = «when I leave» ac = «all the lights... off»
Rephrased Command
Yes
Chatbot
Hi, I'm bot. How can I assist you?
User
I would like all the lights in the house to be switched off when I leave
Chatbot
Ok, the rule is...
Rasa Intent Classifier
Classify the intents and extract the entities
RASA
Rasa NLG
Generate responses for the user
Does the command generate a breakdown?
No
Yes
Are all mandatory parameters present?
No
Ask for missing parameters
Rasa Dialogue Manager

Figure 1: System architecture.

3.1. Rasa

Rasa is a framework for creating chatbots that makes use of machine learning techniques to handle intent recognition, entity extraction and conversational flow management. Rasa provides for the use of "custom actions", which allow Python functions to be executed when the user expresses certain intentions or when certain conditions occur in the conversation.

The basic idea is to use custom actions to fetch the ChatGPT API when specific intents are recognised (an in-depth description is given in the section 3), performing different NLP tasks (section 2.2).

The structure of intents and entities in Rasa should reflect the structure of the smart environment context model. One intent should be defined for each individual trigger or action in the context, each of which will have a set of entities to recognise the parameters required to correctly use the trigger or action into a rule (e.g., the desired state of an appliance, the reference room). Moreover, one intent that identifies if the input is a trigger-action rule (without the need to identify the single components) is needed in order to identify when sending the rule to ChatGPT Splitter for further analysis.

For example, assuming an environment with two triggers (temperature sensor, and motion sensor) and two actions (light status and sending notifications), the Rasa Intent Classifier supports an intent for the recognition of inputs related to temperature control, one intent for the inputs related to the presence of motion in a room, one intent to handle inputs for turning lights on and off and one intent for classifying input about receiving notifications. The entity recognition will make it possible to extract the relevant parameters from the inputs received, such as the number representing the temperature, the state to be set for the lights (e.g., on/off), the room in which to control the temperature or the lights, and so on.

This structure is required since the Rasa model will manage complex rules "part by part", using the chunks extracted by ChatGPT Splitter representing the single triggers and actions involved. This design requires less effort to add, remove or modify any trigger or action of the context since every intent (thus, every trigger and action in the context) is independent from the others.

3.2. Prompting ChatGPT

The system designed uses ChatGPT to complete different tasks by using predefined prompt templates that includes the instruction to be followed and, in some case, a set of examples (composed by question-answer pairs) to show ChatGPT how to achieve the final goal (few-shot learning) [2].

In particular, the following are the tasks that can be executed using ChatGPT. Each of these requires a different prompt to work properly:

- **Splitting complex rules into simple commands:** when the user describes how the smart home should act, the sentence produced is usually composed almost by a trigger and an action. This component should split those sentences into single commands (e.g., “I want to be notified if someone enter home while I’m outside” → “I want to be notified”, “if someone enter home”, “while I’m outside”) to be sent to the Rasa Intent Classifier;
- **Managing breakdowns during rule creation:** when the user describes a rule, it is possible that one or more trigger and action are not recognized by the Rasa Intent Classifier, generating a so called “conversational breakdown”. In this case, the breakdown can be due to **two causes**: the trigger/action is not recognised because of a lack of understanding of the intent classifier, or the trigger/action is not present in the context, thus is not executable by the system nor recognisable by the classifier. When a breakdown occurs, the input is sent to ChatGPT Classifier along with a description of the current context, asking to rephrase and classify the input if it falls in a trigger or action possible command (covering the lack of understanding of the Rasa Intent Classifier), or to classify the input as “unavailable” and to generate an explanation message for the user (covers the case of the missing trigger or action in context). Using this technique, the breakdown message will not be the classic 'Sorry, I didn't understand' but will provide precise information on why the request failed;
- **Question-answering:** used to fulfil information requests about the context, about the triggers and the actions, or provide explanation about the rules created. The system relies on a knowledge base that stores information as text embeddings. When a user submits a request, the system converts the query into an embedding and applies cosine similarity [5] to identify and extract the relevant embeddings from the knowledge base. These embeddings are then converted back into plain text and fed to ChatGPT to generate a summary message for the user.

The mentioned tasks could be performed also using specific trained models for each task, but this would require a large amount of training data, moreover the adaptation to different context would require training different models and different training data for the same task.

4. Example Scenario

The system provides for the creation of customisation rules using written natural language both in English and Italian and it is designed to understand and manage complex input that describes rules composed by multiple triggers and actions, describing how the smart environment should behave. During the creation of the rule the user can ask requests for explanation or information about the environment, the triggers and the actions present in the context. In general, all user inputs are first classified using the Rasa Intent Classifier and, depending on the resulting intent, the corresponding actions are executed (which may involve a default static response or the execution of Python code for ChatGPT API calls).

In an example interaction scenario the user sends a single input for describing the desired smart home behaviour, thus putting together all the triggers and actions that should compose the rule (e.g., “Turn off the lights and lock the main door if it is after 11 p.m. and I’m in bed”).

If the Rasa Classifier associates the user message with a trigger-action rule intent, then the input is sent to the ChatGPT Splitter component that performs the required task and returns an array of strings representing the single triggers and actions present in the input (in the example, triggers = [“if it’s after 11 p.m.”, “I’m in bed”]; actions = [“turn off the lights”, “lock the main door”]).

Then, *each element* of the resulting array is sent individually to the Rasa Classifier to be mapped into a trigger or an action, to extract the entities already present and checking for required but missing entities to be asked to the user, starting the so called “slot-filling” phase (in the example, triggers = [time_schedule(“after”, “11pm”), bed_presence(“inside”)]; actions = [light_state(“off”, *room required*), *not_classified*(“lock the main door”)].

If one or multiple elements of the array cannot be classified as trigger or action, thus it causes a breakdown, and they are sent to the ChatGPT Breakdown Manager (in the example the action “lock the main door”). This component can return the rephrased input along with the possible intent of the trigger or action (without user intervention), otherwise it can generate and return an explanation about why that input cannot be classified as a valid trigger or action, and eventually the most similar trigger or action to replace it with, using the information and instruction given in the prompt (e.g., “I have no control over the lock, but I can activate the alarm instead”). In the latter case, the user can choose between deleting the rule, continue the rule without the unrecognized element or replace the unrecognized element. If the user chooses to replace the element, the new input follows again the steps above until the user is satisfied with the rule.

At this point, the system starts the slot-filling process asking the user for the missing – and mandatory – parameters, needed to correctly build the rule. In the example, the system will ask for the room in which the lights are to be switched off.

After collecting all the necessary parameters, the chatbot sends a message with a summary of the rule created and asks the user whether he/she wants to save the rule or delete it.

Finally, the rule is saved and associated with a name chosen by the user.

5. Conclusions

In this paper, we propose a novel approach for building a conversational agent that combines the capabilities of ChatGPT and Rasa Open Source for creating trigger-action rules and controlling smart objects inside smart environments. Our approach integrates ChatGPT's abilities in natural language processing through prompting and few-shot learning and Rasa Open Source's features in handling intents, entities, forms, and executing actions. We designed custom Rasa actions that invoke ChatGPT's API to process complex customization rules, manage conversational breakdowns, and answer questions about the smart environment. Thus, the proposed tool allows for the creation of complex trigger-action rules that involves several triggers and actions. Overall, this hybrid solution exploits the significant capabilities of LLMs while mitigating their risks by integrating them with trustworthy tools for greater security and control.

6. References

[1] Barbara Rita Barricelli, Daniela Fogli, Letizia Iemmolo, and Angela Locoro. 2022. A Multi-Modal Approach to Creating Routines for Smart Speakers. In Proceedings of the 2022 International Conference on Advanced Visual Interfaces (AVI 2022), Association for Computing Machinery, New York, NY, USA. DOI:https://doi.org/10.1145/3531073.3531168

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. DOI:http://arxiv.org/abs/2005.14165

[3] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2020. HeyTAP: Bridging the Gaps Between Users’ Needs and Technology in IF-THEN Rules via Conversation. In Proceedings of the International Conference on Advanced Visual Interfaces, ACM, Salerno Italy, 1–9. DOI:https://doi.org/10.1145/3399715.3399905

[4] Simone Gallo and Fabio Paterno. 2022. A Conversational Agent for Creating Flexible Daily Automation. In Proceedings of the 2022 International Conference on Advanced Visual Interfaces (AVI 2022), Association for Computing Machinery, New York, NY, USA. DOI:https://doi.org/10.1145/3531073.3531090

[5] Daniel Jurafsky and James H Jr Martin. Vector Semantics and Embeddings. In Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition (Third Edition draft). Standford Edu. Retrieved from https://web.stanford.edu/~jurafsky/slp3/

[6] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2023. Large Language Models are Zero-Shot Reasoners. DOI:http://arxiv.org/abs/2205.11916

[7] André Sousa Lago, João Pedro Dias, and Hugo Sereno Ferreira. 2021. Managing non-trivial internet-of-things systems with conversational assistants: A prototype and a feasibility experiment. Journal of Computational Science 51, (April 2021), 101324. DOI:https://doi.org/10.1016/j.jocs.2021.101324

[8] OpenAI. 2023. GPT-4 Technical Report. DOI:http://arxiv.org/abs/2303.08774