A Conversational Agent for Creating Flexible Daily Automations

Insert Subtitle Here

FirstName Surname[†]
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

FirstName Surname
Department Name
Institution/University Name
City State Country
email@email.com

ABSTRACT

The spread of sensors and intelligent devices of the Internet of Things and their integration in daily environments are changing the way we interact with some of the most common objects in everyday life. Therefore, there is an evident need to provide non-expert users with the ability to customize in a simple but effective way the behaviour of these devices based on their preferences and habits. This paper presents RuleBot, a conversational agent that uses machine learning and natural language processing techniques to allow end users to create automations according to a flexible implementation of the trigger-action paradigm, and thereby customize the behaviour of devices and sensors using natural language. In particular, the paper describes the design and implementation of RuleBot, and reports on a user test and lessons learnt.

CCS CONCEPTS

• Human-centered Computing → Natural Language Interfaces • Information Systems → Chat

KEYWORDS

Chatbot, Trigger-Action Rules, End-User Development

ACM Reference format:

FirstName Surname, FirstName Surname and FirstName Surname. 2018. Insert Your Title Here: Insert Subtitle Here. In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1. Introduction

The Internet of Things (IoT) has become very pervasive, and current forecasts[1] indicate that such trend will even increase in the near future. In this context it is important to exploit such technological offer through automations that are able to coordinate the behaviour of connected objects and devices. Various artificial intelligence techniques can be useful to automatically identify the most relevant ones. They may obtain mixed results (see for example [19]), because they may generate actions that do not match the real user needs or people may have difficulties in understanding the automatically generated automations. In addition, people have dynamic and specific needs, thus it becomes important to allow them to directly personalize the possible automations, even if they are not professional software developers, according to the End-User Development (EUD) [16] paradigm. In this perspective, the Trigger-Action Programming (TAP) approach seems particularly suitable since does not require specific algorithmic abilities because it mainly allows users to connect the relevant dynamic events detected through sensors or services with the desired actions. Several tools have been put forward both at a research and commercial level (such as IFTTT[2], Node-Red[3], Zipato[4]) to provide support for some kind of TAP. At a research level, various ways to support end users in composing (e.g. [4, 5, 7, 11, 17]) or understanding trigger-action rules have been put forward, such as dynamic recommendations [10] or visual predictions of the future behaviour [6].

Various composition paradigms have been considered to support the development process of trigger-action rules. By composition paradigm we mean how the tailoring environments guide the rule development process, how they present the relevant concepts and interact with users. In general, such composition paradigms have exploited the visual modality supporting data flow representations or wizard-like styles or block-based manipulations. The conversational composition paradigm has received some attention but with limited solutions that have not considered the relevant aspects in TAP, such as the possibility to distinguish between events and conditions, compose multiple triggers and actions, and indicate triggers associated with when some event does not occur. In general, limited support by the tailoring environment can generate some of the potential issues in interpreting the trigger-action rules [3]. For example, it is important to allow users to clearly understand the temporal dimension of the triggers and actions considered [13], since their misunderstanding may lead to undesired effects such as unlocking doors or activating heating systems at the wrong time.

[*]Article Title Footnote needs to be captured as Title Note
[†]Author Footnote to be captured as Author Note

WOODSTOCK'18, June, 2018, El Paso, Texas USA

[1] https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/
[2] https://ifttt.com/home
[3] https://nodered.org/
[4] https://www.zipato.com/

In this paper we present a conversational agent able to support end users to create automations for daily environments, such as the home, in terms of trigger-action rules that allow them to flexibly indicate the desired objects' behaviour. Thus, after discussing related work we introduce the design of the proposed tool, we then detail how we have implemented it, and the user feedback received. Lastly, we draw some conclusions and provide indications for future work.

2. Related work

The possibility to support end-user creation of automations in the trigger-action format is provided by several commercial or research tools. However, usually this is provided with limitations that do not allow users to express all the possible types of rules. For example, Amazon Alexa in addition to the "classic" features, relating to carrying out instant actions such as turning on the lights or starting the music, it provides the possibility of creating "routines" using the graphic interface made available by the relevant smartphone application. Similar functionalities are provided by Google Home. IFTTT is an online service accessible through visual interfaces that offers the possibility to create rules in trigger-action format (called applets) that give rise to simple applications for controlling smart objects or web applications. When using the free version, IFTTT allows the creation of maximum three rules consisting of a single trigger and a single action. While the commercial version supports the possibility to express more structured rules but without clearly indicate the difference between events and conditions. In general, such commercial tools have not considered the conversational style for creating trigger-action rules or when they consider it is only for some limited types of rules.

Some research studies have started to consider the conversational approach. HeyTAP is a conversational agent for personalizing the behaviour of house smart objects [8]. It presents a multimodal interface through which the user can express preferences related to the functioning of the installed devices. HeyTAP requires the trigger part of the rule to be defined individually and separately from the action to be performed. Moreover, it does not allow the direct creation of the rule but requires further interaction with a "classic" interface to choose the routine, among those proposed by the system, that is closest to the user's request.

Although not particularly focused on the Internet of Things, SUGILITE [15] uses the programming by demonstration paradigm for the creation of automations on mobile devices. The users can associate a set of actions with a custom voice command, showing how these actions are to be performed through direct navigation. Another relevant work is InstructableCrowd [12], a framework which enables users to converse with the crowd through their phone and describe a problem. Then, it provides a graphical interface for crowd workers to both chat with the user, and compose a rule with a part connected to the user's phone sensors, and a part to its effectors. However, in IntructableCrowd the rules are created through a graphical interface.

CAPIRCI [2] is a multi-modal interactive system for off-line collaborative robot programming. The approach provides the users with two different but integrated ways for defining robot tasks: the former, based on natural language processing, should be firstly used to address the programming problem, by obtaining the specification of a simple task; the latter, based on a component-based visual language, can be used to refine the program. Valtolina et al. [18] reported on a study evaluating the benefits of a chatbot in comparison to traditional GUI, specifically for users with poor aptitude in using technologies. They considered two example scenarios in the healthcare and smart home fields, and found that for the user experience the chatbot application appears to be better than the GUI-based one. One further contribution [1] investigated the effects of including a conversational agent for helping end-user developers in defining the behaviour of point-and-click games through event-condition action rules. The comparison of the versions with and without the chatbot showed a decrease in the perceived cognitive load in complex tasks when using the chatbot. Despite such promising indications, the conversational interaction style has received limited attention for supporting creation of trigger-action rules in general, and in this paper we present RuleBot a proposal aiming to contribute to cover such gap.

One work that addresses similar issues is Jarvis [14]. It allows users to create rules for the instantaneous or delayed execution of commands on devices inside the home, with a limit of one trigger and one action. Even if it supports the execution of instant actions (e.g. "turn off the lights or play some music") and causality queries (e.g. "why did the lights turn on in the kitchen?"), it does not allow users to have a full and detailed control of trigger-action rules. Thus, in Jarvis it is not possible, for example, to specify rules with actions that require a certain delay and, at the same time, some condition to be verified in order to be triggered (e.g. "turn on the light in the living room at 7:00 pm if it's dark"). This kind of rules can be created using RuleBot, since the implemented support allows the creation of rules containing multiple triggers and actions. Other aspects supported by RuleBot but not Jarvis concern the possibility to manage triggers with negation (e.g. when I don't take the medicine"), and the possibility to clearly distinguish between events or conditions when creating a trigger.

3. The Proposed Solution

The purpose of the chatbot is to be able to perform conversations with end users aimed at creating trigger-action rules implementing automations for available connected objects and services, which can then be activated and executed. Considering the issues detected in several previous languages for trigger-action rules we adopted a language that is able to clearly distinguish between events and conditions, and express multiple triggers, which can be composed through logical operators (OR, AND); and it can support multiple actions, which can be composed sequentially; and the possibility to trigger a rule when some event does *not* occur. This flexibility of the language has implications in the design of the conversational agent that has to allow the users to specify all the relevant information. In addition, the possible triggers are logically organised according to three main dimensions: *user*, aspects related to their emotional, and physical state, and the activities that they perform; *environment*, aspects related to the surrounding elements, such as light, noise, temperature, humidity, and associated services

such as weather forecasts; and *technology*, related to the state of the various devices available, such as TV, smartphones, PCs, and their services. The possible actions can change the state of the appliances and devices available, can generate reminders and alarms through various channels, and can activate device dependent services, such as those associated with Alexa.

We followed an iterative design process in developing the proposed solution for a conversational agent able to support such language. In the first version (V.1) to achieve greater accuracy in recognizing the user's intention, it was decided to create a specific intent for each possible trigger and action present in the language (66 intents were created for triggers and actions, 6 for rule creation support features such as saving and deleting rules, or providing information about its functionalities). For each intent, the set of training phrases included phrases that referred directly to the single element considered. For example, the intent for the recognition of inputs related to a motion sensor was trained with phrases such as "if the motion sensor in the kitchen is active", or "if the motion sensor is activated in the bedroom". This structure of intents and entities, however, presented limitations in the interaction between the chatbot and the user. In fact, to build the rule, each user input can include at most one trigger or action, making the conversation fragmented and unnatural when users want create rules with multiple triggers and actions. In addition, the training phrases used had a "device-centric" level of abstraction [9] (e.g. "when the motion sensor is the kitchen becomes active" or "if the thermostat detects a temperature lower than 10 degrees"), for which the user inputs must contain a direct reference to the sensor to be used, forcing the users to conform to this specific way of looking at automations. A first user test of the initial prototype was carried out with twenty-three students of a course of a Digital Humanities degree. They had to specify five rules with increasing level of complexity. Overall, they did not much like the composition style exploited in that chatbot prototype, which was judged somewhat imprecise, requiring several interactions, thus not very efficient, with some features requiring improvements.

Then, we considered an approach based on intents that can categorise multiple triggers or actions in one sentence by providing training phrases containing, for example, one trigger and one action, two triggers and two actions. The organisation of the intents in this version (V.2) no longer followed the structure of the previous one (one intent for each single possible trigger or action), but referred to the possibility of intents associated with multiple triggers and/or actions within the user input. Initially, three main intents were considered, which correspond to inputs consisting of: two triggers, two actions, and one trigger and one action. So, for example, we used training phrases such as "turn off the lights when I'm leaving home" or "send me a notification if it will rain tomorrow" to categorise one trigger and one action. Training phrases such as "if it's raining or snowing" would be associated to the intent that categorises two triggers, and "turn on the light and play some music" for the intent to categorise two actions. However, this approach raised several problems regarding: the quantity and the quality of the training phrases; the recognition of the specific trigger and action present in one input; the scalability of the corresponding architecture. The first point concerns the difficulty of managing all the possible instances of the various possible combinations (trigger + trigger, action + action and trigger + action) and the respective intents. To get an idea of the possible combinations, consider the number of triggers and actions types in the language (at the time of writing there are 58 and 8 respectively). In the worst case, represented by the combination of two triggers, we obtain exactly 1653 combinations, and then we need to consider the possible variants for each element (for example, the verification of the activation of the smoke sensor could be expressed with sentences such as "if the smoke sensor in the kitchen is active", "if there is smoke in the kitchen", "when smoke is detected in the kitchen" and so on). The second problem concerns the identification of sensors and/or actions within the user input. In fact, even if an intent can recognise that the input contains, for example, one trigger and one action, it is then problematic to identify exactly which sensors or devices are referred to. Finally, the scalability of the system is compromised because the addition of even one sensor to the language would require writing many training phrases considering the new sensor in association with all the existing ones.

We thus designed and developed a new solution (V.3) with the aim of overcoming the limitations of the previous ones. The new system is based on the combined use of two different components: one dedicated to receiving inputs and sending responses with the user, called "**Dialogue Interface**"; another, to which the appropriately processed input is submitted, which deals with the classification of the intents and the extraction of the parameters, called "**Intent Classifier**". A "**Dialogue Manager"**, implemented through a webhook server, is interposed between these two components, and processes the inputs received from the former and generates the responses to be sent back based on the data received from the Intent Classifier (see Figure 1). This process, explained in detail in the next section, allows the reception of even complex inputs, containing multiple triggers and actions at the same time. In addition, the sets of training sentences have been appropriately expanded with sentences that consider different syntactic constructions as well as different communication styles, like sentences that use "people-centric" and "info-centric" abstractions [6] to refer to a sensor or a device (e.g. "when I enter the bedroom" rather than "when the motion sensor in the bedroom becomes active").

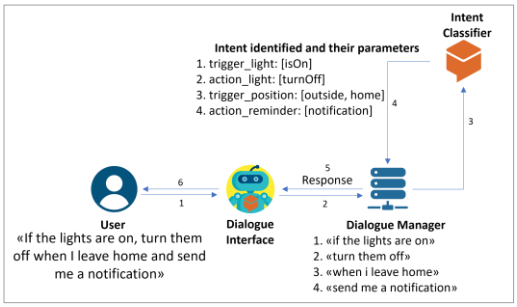

Figure 1. RuleBot architecture

4. Implementation

Dialogue Manager. The Dialogue Manager is implemented through a webhook component running on Node.js, and is essential for managing all the chatbot features. It receives the inputs from the Dialogue Interface and the related intents from the associated Intent Classifier, validates and processes the extracted parameters, manages the creation, updating and deletion of contexts, and carries out a set of secondary functions, such as the management of saving and deleting rules.

When the Dialogue Interface receives a complex sentence containing more triggers and actions as input, it sends it to the Dialogue Manager, which breaks it down into smaller and simpler sentences that represent the single triggers/actions involved. Thus, a list of multiple sentences is obtained from a single complex one, and for each sentence obtained, a call is made to the Intent Classifier. For each of the phrases received this latter performs the classification of the intents, and identification of the relative parameters, and then returns a response in JSON format containing the extracted information to the Dialogue Manager.

The subdivision of the initial sentence provided by the user into a list of smaller sentences has been implemented through an algorithm that uses regular expressions and a Parts of Speech (PoS) Tagger[5]. In particular, regular expressions search within the user input sentence for some specific words and characters that indicate the border point between multiple triggers with high probability. The terms that support identification of the triggers are "when", "if" and "while", in addition, the conjunctions "and" and "or", and punctuation marks such as commas and periods are also taken into consideration. Regarding the actions, their identification is based on the presence (through the data extracted from the PoS tagger) of verbs that express actions such as "turn on", "turn off", "send", "play", "open", "close" and so on. Specifically, the verbs are taken into consideration when they are[6]: in the second person singular, present tense of the indicative mood; in the second person singular of the imperative mood; infinitives.

After receiving the corresponding intents and related parameters from the Intent Classifier, a verification process is initiated to identify any missing parameters to successfully create the rule. For this purpose, two reference models have been defined: one for describing triggers and one for actions. Both templates (defined in JSON) consist of "name intent: list of required parameters" pairs. For each identified intent, the Dialogue Manager performs a comparison between the extracted parameters and the mandatory parameters, then a queue is created containing the missing parameters. Subsequently, the chatbot asks the user to provide the missing data, updating the queue as the requests are satisfied.

The generation of responses has been divided into two categories: predefined and template-based responses. The former, used for requesting parameters, have been statically defined within a JSON file. Each trigger or action (identified by the name of the corresponding intent) is associated with the list of mandatory parameters; for each parameter there are two types of responses: one for asking for the missing parameters and one for indicating that the input parameter provided was of the wrong type.

The template-based responses are used in the final part of the rule creation process. When the rule is complete, the agent reconstructs it in natural language starting with the set of parameters that form it. For this purpose, a set of templates have been defined that "translate" and concatenate the parameters in a meaningful sentence. This type of response is used to provide feedback to the user in order to verify the correct understanding and creation of the rule by the chatbot.

Dialogue Interface. This component manages the user inputs, which can be of two types: new rules or parameters requested by the chatbot to complete a rule under editing. Inputs to complete a rule are identified because they are entered only when this component is in specific contexts, in the other cases the input is associated with the intent "Rule Identifier", and, after appropriate processing by the Dialogue Manager, is subjected to recognition by the Intent Classifier.

The "prompt" and "get" intents manage the insertion of missing data for completion of the rule (i.e. the mandatory entities not yet present in the input rule). In particular, the "prompts", which are activated through the use of Dialogflow events, are used to activate the message requesting the missing data (e.g. "In which room do you want to turn on the lights?"), and set the contexts necessary to receive the response. The user response is captured by the corresponding "get" intent or, in the case of unexpected responses, by the associate fallback intent. For each type of entity there are therefore the intents "prompt", "get" and "fallback". In addition to classic entities, we used composite entities: "triggerValue", which groups all the entities that refer to the possible values that can be associated with a trigger (e.g. numbers, dates, weather conditions, cognitive and emotional states and so on); "actionValue", which refers to the values that can be associated with the actions (e.g. the colours of the lights or the text to be associated with the reminders). Their use is linked to intents of type "get"; in fact, the respective "get.triggerValue" and "get.actionValue" make use of these entities to recognize the corresponding terms.

Therefore, referring to the intents of type "prompt", "get" and "fallback", only the "get" intents contain training phrases because those of the "prompt" type correspond to a support function and do not need a user input, while the "fallback" ones are automatically activated when the input does not match the correspondent "get" intent.

Some training phrases (between 10 and 20) have been used for each "get" intent. For example, the intent "get.binaryState" contains training phrases such as "is open", "is closed", "activates", "deactivates", "switch on", "switch off", associated with the entity "BinaryState". The intent "get.notificationMode" (which intercepts the answer to the question "Do you want to send a notification, an email, a voice message or an SMS?") has among its training phrases: "send an email", "send an sms", "a notification" and so on.

[5] The Python NLP package called "Stanza" by Stanford University. https://stanfordnlp.github.io/stanza/index.html

[6] Currently the chatbot supports Italian.

We have used Dialogflow contexts for directing the flow of the conversation by limiting the intents that can be activated upon receipt of a certain input, and also for tracking and storing the interactions that take place during rule creation. In particular, the two main contexts are: "queue", after receiving the input containing triggers and actions, and after recognizing them through the Intent Classifier, a queue is created containing the missing parameters to ask the user for, and each time the user provides a parameter, the queue is updated to request the next one; "model" keeps track of the recognized intents and related parameters with the extracted values, and is updated in conjunction with the "queue" context. Every time the user provides a missing parameter, the "parameter: value" pair is added to the corresponding intent model. When the queue ends, this context is saved and translated into a data structure that describes the rule created.
Both contexts are generated and managed by the webhook server, and are related to a single rule creation session.

Intent Classifier. This component is responsible for classifying inputs that represent triggers or actions after the decomposition performed by the Dialogue Manager. The classifier associates a single intent for each possible trigger and action defined by the language (66 intents have been created). Thus, for example, it contains intents to classify inputs such as "when the light turns on", "if it's raining", "send me a reminder", "play some music" and so on, which will be respectively associated with the intents "trigger_light", "trigger_weather", "action_reminder" and "action_music". A set of entities (26) has been defined to recognize the terms that identify the required parameters. For instance, the sentence "when the light turns on" contains the entity "triggerType", which determines if the trigger is an event or a condition ("if" or "when"), and the entity "binaryState" to identify state changes ("on" or "off").
For each intent a variable number of training phrases has been defined, considering the intent complexity and the number of entities associated: intents that define triggers contain a number of training phrases ranging from 40 up to 70. The actions are in general simpler than triggers and have fewer required parameters, and thus have received a number of training phrases between 20 and 40.
The training phrases aim to reflect different ways of expressing a concept. Thus, we used phrases that refer directly to a sensor or device (e.g. "If the motion sensor of the kitchen is active"), but also to a person (e.g. "if I enter the kitchen") or to information (e.g. "If there is movement in the kitchen").

Figure 2 shows an example of use, the texts have been translated in English for wider readability. It illustrates the creation of a rule containing two triggers and two actions. In this case, all the desired triggers and actions are in the first user input (Fig. 2a), while the successive interactions are limited to obtaining the necessary information for completion of the rule. Figure 2b shows the next steps followed by the request for the input of the logical operator for the concatenation of the two triggers. The conversation ends with a feedback message from the chatbot that summarizes the rule created, followed by the proposal to save it (see Fig. 2c).

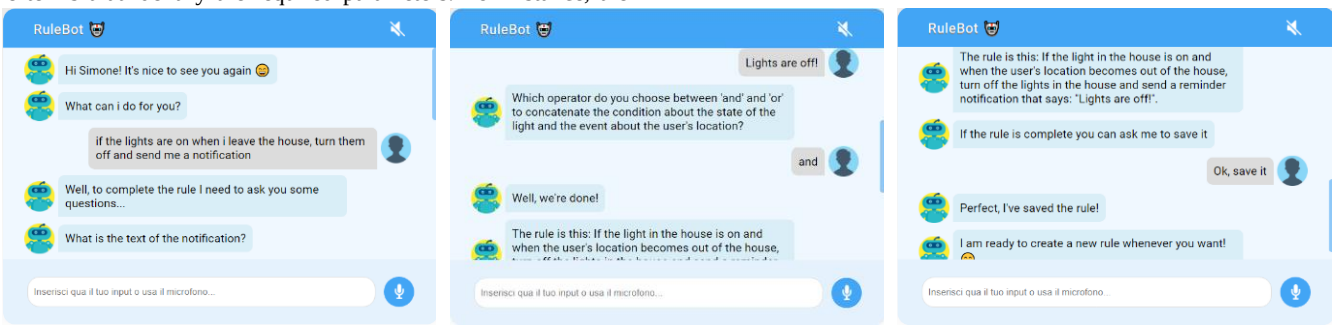

Figure 2a, b, c. Creation of a rule composed of two triggers and two actions with RuleBot

5. User Test

A user test has been carried out on the latest version of the chatbot to better understand its usability. To make the test and user feedback more consistent, exhaustive, and informative, it was decided to define "scenarios" that represent situations in which it would be appropriate to create customization rules.
The scenarios were presented in order of increasing difficulty, requiring the specification of simple rules consisting of one trigger and one action, to more complex rules consisting of two of each. Users were asked to address the proposed scenarios by creating customization rules using two different tools: TAREME and RuleBot. Finally, each user filled out a questionnaire in which they were asked to express their opinions about the efficiency, and their appreciation of the experience with the two tools.
TAREME[7] [7, 17] is a publicly available tool supporting graphical wizard composition of TAP rules. It is visually structured into two main parts, one for triggers and one for actions, and one sidebar providing feedback on the progress achieved in creating the rules. Both the trigger and action parts are organized in terms of main categories, which when selected, unfold their sub-categories, and it is possible to iterate until the basic elements with their attributes are

[7] https://tare.isti.cnr.it/RuleEditor/login

visualised. Depending on the rule elements selected, a sentence appears on the top part of the user interface during the editing process to provide feedback in natural language of the rule created. It applies a wizard-like compositional style, and we chose it also because it is able to support the same rule language as RuleBot, thus the usability evaluation is not affected by issues related to the different ability to express rules between the two considered tools.

5.1 Users and Tasks

We had ten participants: eight had no expertise in programming languages, only two had knowledge of languages such as HTML and CSS; eight males, two females, their ages ranged from 20 to 30, with an average of 27. They were divided into two groups of five. Both groups used TAREME and RuleBot, in inverse order to balance learning effects. After the creation of the rules, they were then asked to rate six statements about the two tools used, on a scale of 1 to 5, indicating respectively full disagreement and full agreement. Users then gave a motivation for each score assigned.

They received the following four scenarios requiring writing corresponding personalisation rules in trigger-action format.
Main context: You are the son/daughter of an older adult who lives alone. The house s/he lives in is equipped with sensors and smart objects that allow checking what happens inside the house (lights, parent's presence in a given room, temperature or acoustic noise in a room etc.) and execute actions as a consequence (turn on lights, send alarms or reminders, start music, …).
Scenario 1 (1 trigger + 1 action): You as the child are concerned that your parent is not eating healthily, and that s/he is resorting to cold foods too often. Therefore, you would like to receive an email whenever your parent is using the microwave oven.
Scenario 2 (2 triggers + 1 action): Sometimes the parent does not notice that s/he keeps the TV volume too high, even in the late evening. To solve the problem, you would like to receive a message on your cell phone when there is too much noise in the living room after 10 pm (30 decibels are exceeded).
Scenario 3 (1 trigger with negation + 2 actions): You want to make sure that your parent correctly takes a certain medicine, Pradaxa, which must be taken by 3 pm. To this end, you would like to create an automation that reminds your parent to take the Pradaxa if s/he has not yet done so. To be effective, it would be helpful to send a notification to the parent's phone, and also to turn on a red light for 1 minute in the kitchen, where your parent is usually at that time.
Scenario 4 (2 triggers + 2 actions): Your parent too often spends the afternoon (after 3:30 pm) in the living room reading the newspaper. The living room is dark, consequently s/he always turns on the lights to read. You would like your parent to receive a notification by cell phone in these situations, something like, "Dad, why don't you go out for a walk?". In addition, the "activating" light scene should also start.

5.2 Task Performance

Time measurement. RuleBot supported more efficient performance than TAREME. On average, to perform all the tasks users took 7.38 minutes with RuleBot, and 11.33 minutes with TAREME. The mean execution time (in minutes) per task are:
Task 1: RuleBot (1.26), TAREME (1.41);
Task 2: RuleBot (1.54), TAREME (1.41);
Task 3: RuleBot (1.57), TAREME (3.55);
Task 4: RuleBot (2.36), TAREME (3.19).

Errors analysis. We checked whether the rules created were well-formed. The errors present were classified according to three categories, and refer to correctness of**:** the trigger or action, the choice between an event or a condition, and the use of the NOT operator for the creation of triggers with negation. In addition, for each category, errors have been classified as severe (scored 1 point) or moderate (scored 0.5). Severe errors are those that completely change the behavior of the rule or lack a required element from the task description. An error is considered moderate if when it occurs the rule does not correspond exactly to the task description, but the behavior obtained is very similar to it. A total of 80 rules were analysed. The total error score in rules created with RuleBot was 27.5, while with TAREME it was 23.5. The overall error score for each task was:
Task 1: RuleBot (1), TAREME (2);
Task 2: RuleBot (9), TAREME (5.5);
Task 3: RuleBot (7.5), TAREME (8.5);
Task 4: RuleBot (9.5), TAREME (7).

The data obtained indicate that the wizard-based tool is less prone to errors than the chatbot. Both the total errors and the overall error scores per task describe greater accuracy in the use of the visual environment over the conversational one.

Errors in triggers and actions: in RuleBot most errors concern the incorrect use of the comparatives of equality, greater or lesser (9 rules out of 40), often associated with triggers on schedules (e.g. user said "if it's 3pm" instead of saying "if it's after 3pm"); 8 rules out of 40 present errors concerning the addition of an extra trigger or the non-insertion of a necessary trigger. Regarding TAREME, 7 rules out of 40 present an unsolicited trigger or the non-insertion of a necessary trigger; only 4 rules contain the incorrect use of comparisons; 3 rules present an incorrect choice of trigger, even if for all three cases the error was considered moderate as they concerned the choice of similar triggers (e.g. the trigger on the brightness in a room is used rather than the one on the state of the light).
Errors in the choice between event and condition: for RuleBot this is the most common type of error: 20 out of 40 rules have moderate errors regarding the choice between event and condition. Only 1 rule has an error classified as severe because it prevents the rule being activated. TAREME instead presents 16 rules out of 40 with a moderate error regarding the choice between event and condition and 1 rule in which the error was considered severe.
Regarding **errors related to the use of the NOT operator,** these only occurred in rules produced with TAREME. In two rules there was a moderate error concerning the specification of the incorrect

time. One rule has a severe error because the negation was added to a trigger for which it was not required.

Conversational turns. The conversational turns have been counted starting from the first user input to describe the rule to be created. The turns count ends when the user sends the message to save the rule (thus considered complete and correct). Fallback messages have been included in the count. On average the number of turns was:
Task 1: 6.2, Task 2: 8.2, Task 3: 7.2, Task 4: 7.1.

In general, there does not appear to be a particular correlation between the complexity of the rule, the time taken to create it, and the number of turns taken. For all the tasks, in fact, most of the interactions refer to the fulfilment phases and, in particular, to the collection of data needed to create the actions associated with reminders (present in all the tasks), which require the presence of several parameters.

5.3 User Feedback

Users assigned scores on a 1 to 5 scale to statements associated with each task and provided associated comments.

Composing one trigger. The scores were assigned to the statements: "I found it easy to edit the "trigger" portion of a rule using the TAREME/RuleBot tool user interface":
TAREME (Min 2, Max 5, Median 4, Mean 3.9),
RuleBot (Min 2, Max 5, Median 5, Mean 4.4).
As the scores show, for both tools, users had no particular difficulty in creating a single trigger. For TAREME, although the comments highlight good usability and intuitiveness, some express difficulty in moving between the different categories of triggers and actions. The comments about RuleBot refer to the ease of use given by the possibility of expressing oneself in natural language, and of producing commands independently without having to be limited by rigid categories.

Composing two triggers. The scores were assigned to the statements: "I found it easy to compose two "triggers" of a rule using the TAREME/RuleBot tool user interface:
TAREME (Min 2, Max 5, Median 4, Mean 3.7),
RuleBot (Min 2, Max 5, Median 5, Mean 4.3).
As mentioned, the composition of two triggers is done through the use of the logical operators (AND/OR). Also in this case for TAREME the comments focused on the increasing difficulty in identifying the correct trigger among the various categories. Thus, the time spent and the attention required increased. No particular problem was reported regarding the specification of the logical operators, which were immediately identified and easily used by all users. In this case RuleBot was preferred over TAREME for the possibility of specifying the operators to be used directly in the initial input, concatenating multiple events and/or conditions using the conjunctions "and" or "or". In addition, it was emphasized how the greater freedom in the production of the rule makes the task faster and more intuitive, especially with an increasing number of triggers.

Use of "NOT" operator. The scores were assigned to the statements: "I found it easy to edit a "trigger" that uses the "not" operator using the TAREME/RuleBot tool user interface":
TAREME (Min 2, Max 5, Median 4, Mean 3.8),
RuleBot (Min 2, Max 5, Median 5, Mean 4.3).
The performance of this task received mixed comments. For both tools some users had no particular difficulty, while some others needed longer time to understand the correct functioning. In particular, TAREME requires the selection of a box named "not", which activates an additional button named "when". When the "when" button is clicked, a window opens in which the user is asked to enter the start and end time of the negation using text fields. Although this dynamic behaviour, was intuitive for the majority of users (7 users), the remaining found it cumbersome, as they did not fully understand the operation and the need to enter the start and end times of the negation. Less critical comments were provided for RuleBot, which simply requires the user to add the term "not" for entering the requested negation. If no start or end time is specified, the chatbot asks the user what time the missing event or condition needs to be verified. Only one user pointed out the difficulty in producing the correct command for RuleBot to understand. The rest of the users found it natural and obvious.

Composing one action. The scores were assigned to the statements: "I found it easy to edit the action portion of a rule using the TAREME/RuleBot tool user interface":
TAREME (Min 3, Max 5, Median 4, Mean 3.9),
RuleBot (Min 2, Max 5, Median 5, Mean 4.4).
A couple of users found difficulty in finding the actions on the lights; five users misunderstood some items during the specification of the actions on the reminders. These require the inclusion of a recipient depending on the type of reminder chosen: a phone number in the case of SMS reminders, an e-mail address in the case of e-mail reminders. In this case, users entered "my father" or "to me" as the recipient, assuming that the software would automatically figure out how to send the reminder to them. In the case of RuleBot, no particular problems were highlighted; moreover, the chatbot itself asks which email or phone number to send the reminder to, thereby eliminating the possibility of errors.

Composing two actions. The scores were assigned to the statement: "I found it simple to compose two actions within a rule using the TAREME/RuleBot tool user interface":
TAREME (Min 2, Max 5, Median 4, Mean 3.9),
RuleBot (Min 2, Max 5, Median 5, Mean 4.3).
Comments consistent with those for composing an action were provided for both tools. Problems in finding the right elements to select in TAREME and quicker composition time for RuleBot were highlighted. Some additional comments were provided for TAREME: one user had problems because, considering what he had to do to concatenate two triggers by using logical operators, he

could not understand if the same procedure was necessary also for the concatenation of two actions.

Perceived speed and efficiency in creating the rule. The scores were assigned to the statements: "The way rules are built using the TAREME/RuleBot tool is fast and efficient":
TAREME (Min 3, Max 5, Median 3, Mean 3.5),
RuleBot (Min 3, Max 5, Median 4, Mean 4.3).
For TAREME users agree on the difficulty employed in finding the right triggers and actions required to complete tasks, and the presence of ambiguity in the different categories. This can generate uncertainties, and in some cases errors, in selecting the right items, and as a result the time spent and perceived difficulty can increase as the number of triggers and actions required to complete the rule increases. On the other hand, after having found the right element, the possibility to have an overall view of the parameters that make up the trigger and the action, and the possibility to verify the correctness thanks to the displayed natural language explanation of the composed rule, make the tool efficient.
RuleBot was assessed, overall, as faster, but in some cases less precise than TAREME in determining some elements of the rule to create. While the ability to express oneself in natural language makes the interaction more immediate, problems in understanding some terms were highlighted. In some cases, this problem led to the generation of imperfect rules, in others to having to repeat parts of sentences already expressed in the initial input, but requested again during rule completion.

6. Discussion

Although the scores assigned to both tools indicate good usability, the use of the conversational interface received better scores than the classic wizard style interface. In particular, the creation of multiple triggers, the use of the NOT operator, and the efficiency of the system, are those in which RuleBot received clearly better assessments.
Finally, each user was asked which of the two tools they would prefer to adopt and use in their home. Nine out of ten expressed their preference for RuleBot, while one user preferred TAREME. In particular, this user preferred the use of a classical visual interface declaring that it is less liable to generate misunderstandings, especially in reference to long and complex rules.
In general, users found RuleBot faster and easier to use than TAREME. On the other hand, TAREME proved to be more accurate overall due to its ability to keep track of the rule creation steps showing the user the triggers and actions included in the rule in real time. The time spent and frustration experienced - particularly when searching for the right triggers - negatively affected user feedback.
As far as RuleBot is concerned, the problems highlighted by some user regard some uncertainties during the first interactions in figuring out how to formulate the initial sentence containing the triggers and actions, and the non-understanding of some input parts. The first problem does not occur if the user, after logging in for the first time, answers positively to the initial chatbot question "Do you want to know what I can do?". In this case, in addition to an explanation of the functionality, some input examples describing customization rules are provided.
From the analysis carried out it can be seen that RuleBot is faster and more intuitive in creating rules but at the same time it can lead users to make more mistakes than TAREME. The time spent on rule creation with TAREME is proportional to the difficulty of the task to be performed, while with RuleBot the variation in time does not follow such a well-defined trend. In particular, Task 3 with TAREME took the longest times due to the presence of the NOT operator, whose mechanism for entering the start and end times of negation does not seem to be immediate for users.
In general, referring to some user comments, most of the time used for rule creation with TAREME is spent in searching for the right trigger/action among the different available categories. After having selected the right trigger, the error analysis shows that TAREME is more precise when setting the parameters of the chosen trigger (e.g. the choice between event and condition and between the different operators). The ability to try different values and combinations, and at the same time visualize the explanation in natural language provides the chance to notice and correct errors more easily than with RuleBot, which provides feedback on the trigger-action rule only at the end of the creation process.
In particular, the errors concerning the choice between event and condition can be attributed to users' limited understanding of their difference, since RuleBot is well trained in understanding the terms that identify events and conditions. Errors regarding the choice of operator or the absence of required triggers or actions can refer not only to misunderstandings on the part of the user but also on the part of RuleBot, which may fail to fully understand the input received.

7. Conclusions and Future Work

In this paper we present the design and development of a conversational agent dedicated to supporting the end user in creating automations in environments populated by sensors and smart objects. We have analysed different possibilities and arrived at a solution able to allow users the specification of flexible automations in trigger-action format with the ability to manage multiple triggers and actions, and clearly distinguish between events and conditions. As a final result, the chatbot is able to perform a conversation with the user thanks to the use of machine learning and natural language processing techniques, and correctly manage the creation of personalization rules even starting from complex inputs.
In future work we want to address the implementation of a multimodal interface that allows for conversations in speech, integrating it with devices such as Alexa and Google Home; adding functionalities that support causality queries (e.g. "Why did the light turn on in the living room?"); and develop a recommendation system that can propose personalized rules to the user based on the use of devices and sensors.

REFERENCES

[1] Luca Asunis, Vittoria Frau, Riccardo Macis, Chiara Pireddu, and Lucio Davide Spano. 2021. PAC-Bot: Writing Text Messages for Developing Point-and-Click Games. In End-User Development, Daniela Fogli, Daniel Tetteroo, Barbara Rita Barricelli, Simone Borsci, Panos Markopoulos and George A. Papadopoulos (eds.). Springer International Publishing, Cham, 213–221. https://doi.org/10.1007/978-3-030-79840-6_15

[2] Sara Beschi, Daniela Fogli, and Fabio Tampalini. 2019. CAPIRCI: A Multi-modal System for Collaborative Robot Programming. In End-User Development, Alessio Malizia, Stefano Valtolina, Anders Morch, Alan Serrano and Andrew Stratton (eds.). Springer International Publishing, Cham, 51–66. https://doi.org/10.1007/978-3-030-24781-2_4

[3] Will Brackenbury, Abhimanyu Deora, Jillian Ritchey, Jason Vallee, Weijia He, Guan Wang, Michael L. Littman, and Blase Ur. 2019. How Users Interpret Bugs in Trigger-Action Programming. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 1–12. https://doi.org/10.1145/3290605.3300782

[4] Federico Cabitza, Daniela Fogli, Rosa Lanzilotti, and Antonio Piccinno. 2017. Rule-based tools for the configuration of ambient intelligence systems: a comparative user study. Multimedia Tools and Applications 76, 4: 5221–5241. https://doi.org/10.1007/s11042-016-3511-2

[5] Danilo Caivano, Daniela Fogli, Rosa Lanzilotti, Antonio Piccinno, and Fabio Cassano. 2018. Supporting end users to control their smart home: design implications from a literature review and an empirical investigation. Journal of Systems and Software 144: 295–313. https://doi.org/10.1016/j.jss.2018.06.035

[6] Sven Coppers, Davy Vanacken, and Kris Luyten. 2020. FORTNIoT: Intelligible Predictions to Improve User Understanding of Smart Home Behavior. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 4, 4: 1–24. https://doi.org/10.1145/3432225

[7] Luca Corcella, Marco Manca, Jan Egil Nordvik, Fabio Paternò, Anne-Marthe Sanders, and Carmen Santoro. 2019. Enabling personalisation of remote elderly assistance. Multimedia Tools and Applications 78, 15: 21557–21583. https://doi.org/10.1007/s11042-019-7449-z

[8] 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, 1–9. https://doi.org/10.1145/3399715.3399905

[9] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2021. Devices, Information, and People: Abstracting the Internet of Things for End-User Personalization. In End-User Development, Daniela Fogli, Daniel Tetteroo, Barbara Rita Barricelli, Simone Borsci, Panos Markopoulos and George A. Papadopoulos (eds.). Springer International Publishing, Cham, 71–86. https://doi.org/10.1007/978-3-030-79840-6_5

[10] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2020. TAPrec: supporting the composition of trigger-action rules through dynamic recommendations. In Proceedings of the 25th International Conference on Intelligent User Interfaces, 579–588. https://doi.org/10.1145/3377325.3377499

[11] Giuseppe Desolda, Carmelo Ardito, and Maristella Matera. 2017. Empowering End Users to Customize their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools. ACM Transactions on Computer-Human Interaction 24, 2: 1–52. https://doi.org/10.1145/3057859

[12] Ting-Hao K. Huang, Amos Azaria, Oscar J. Romero, and Jeffrey P. Bigham. 2019. InstructableCrowd: Creating IF-THEN Rules for Smartphones via Conversations with the Crowd. Human Computation 6: 113–146. https://doi.org/10.15346/hc.v6i1.7

[13] Justin Huang and Maya Cakmak. 2015. Supporting mental model accuracy in trigger-action programming. In Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15, 215–225. https://doi.org/10.1145/2750858.2805830

[14] 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: 101324. https://doi.org/10.1016/j.jocs.2021.101324

[15] Toby Jia-Jun Li, Amos Azaria, and Brad A. Myers. 2017. SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, 6038–6049. https://doi.org/10.1145/3025453.3025483

[16] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. End-User Development: An Emerging Paradigm. In End User Development, Henry Lieberman, Fabio Paternò and Volker Wulf (eds.). Springer Netherlands, Dordrecht, 1–8. https://doi.org/10.1007/1-4020-5386-X_1

[17] Marco Manca, Fabio Paternò, and Carmen Santoro. 2021. Remote monitoring of end-user created automations in field trials. Journal of Ambient Intelligence and Humanized Computing. https://doi.org/10.1007/s12652-021-03239-0

[18] Stefano Valtolina, Barbara Rita Barricelli, and Serena Di Gaetano. 2020. Communicability of traditional interfaces VS chatbots in healthcare and smart home domains. Behaviour & Information Technology 39, 1: 108–132. https://doi.org/10.1080/0144929X.2019.1637025

[19] Rayoung Yang and Mark W. Newman. 2013. Learning from a learning thermostat: lessons for intelligent systems for the home. In Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing, 93–102. https://doi.org/10.1145/2493432.2493489