Managing Non-Trivial Internet-of-Things Systems with Conversational Assistants: A Prototype and a Feasibility Experiment

André Sousa Lago[a], João Pedro Dias[a,b], Hugo Sereno Ferreira[a,b]

[a] *DEI, Faculty of Engineering, University of Porto, Porto, Portugal*
[b] *INESC TEC, Porto, Portugal*

Abstract

Internet-of-Things has reshaped the way people interact with their surroundings and automatize the once manual actions. In a smart home, controlling the Internet-connected lights is as simple as speaking to a nearby conversational assistant. However, specifying interaction rules, such as making the lamp turn on at specific times or when someone enters the space is not a straightforward task. The complexity of doing such increases as the number and variety of devices increases, along with the number of household members. Thus, managing such systems becomes a problem, including finding out why something has happened. This issue lead to the birth of several low-code development solutions that allow users to define rules to their systems, at the cost of discarding the easiness and accessibility of voice interaction. In this paper we extend the previous published work on Jarvis [1], a conversational interface to manage IoT systems that attempts to address these issues by allowing users to specify time-based rules, use contextual awareness for more natural interactions, provide event management and support causality queries. A proof-of-concept is presented, detailing its architecture and natural language processing capabilities. A feasibility experiment was carried with mostly non-technical participants, providing evidence that Jarvis is intuitive enough to be used by common end-users, with participants showcasing an overall preference by conversational assistants over visual low-code solutions.

Keywords: Internet-of-Things, Conversational Assistants, Software Engineering, Natural Language Processing, Visual Programming

1. Introduction

The Internet-of-Things (IoT) is usually defined as the networked connection of everyday objects with actuating and sensing capabilities, often equipped with a collective sense of intelligence [2]. The integration of such objects creates a vast array of distributed systems that can interact with both the environment and the human beings around them, in a lot of different ways [2]. This flexibility of IoT systems has enabled their use across many different product areas and markets including, but not limited to: personal everyday-carry devices such as smartwatches that can watch over health indicators [3], wide-area monitoring systems that can watch for wildfires [4] or environmental conditions [5], and the several kinds of smart-spaces that have been outspreading, such as smart homes and smart farming [6].

Amongst those, one of the most visible areas of application of IoT is *customized smart spaces*, such as *smart homes*, as the current technology makes it possible for consumers to create a customized IoT experience based on *off-the-shelf* products [7]. The initial popularity of devices such as single-board computers and low-cost micro-controllers, followed by widespread cloud-based solutions controlled by mobile phones, it is now commonplace to remotely interact with a myriad of devices to perform automated tasks such as turning the lights on and opening the garage door just before one arrives home [7, 8]. However, as the number of devices and interactions grows, so does the management needs (and management complexity) of the system as a whole, as it becomes essential to understand and modify the way they (co)operate. In the literature, this capability commonly known as *end-user programming* [9], and once we discard trained system integrators and developers, two common approaches emerge, low-code visual programming solutions and conversational assistants [8].

Visual programming solutions are usually used as centralized orchestrators, with access to all the devices and components that comprise such systems. These can

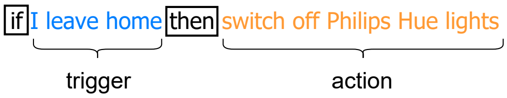

Figure 1: Example of a trigger-action rule for turning off the lights (action) whenever the user leaves the house (trigger).

be *if-then* rules programming solutions[1] such as IFTTT (If This Then That) and Zapier [10], where rules are defined as a sequence of trigger-action *flows*, as exemplified in Fig. 1.

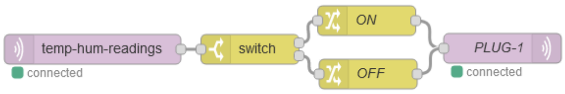

Figure 2: Example of Node-RED *flow*, where the status of a electric plug (PLUG-1) changes (ON/OFF) depending on the current temperature value (SWITCH), provided by the temperature and humidity sensor (TEMP-HUM-READINGS).

More advance solutions exist, such as Node-RED, providing an exhaustive graphical interface through which one can visualize, configure and customize the devices and systems' behaviour [11, 12, 13]. Node-RED provides an programming *canvas* through which users can create, edit and delete system rules and connections in an interface that displays rules and connections as a flow of information, events or action by *drag-n-drop* building blocks (*nodes* and *links*) which are made available through an extensive and extensible *node* palette, as exemplified in Fig. 2. Most visual approaches offer integration with third-party components and services (*e.g.*, calendars and weather services), enabling its use as part of the system's behavioural rules. However, these solutions, in resemblance to workflow-based solutions, have several limitations in terms of dealing with high-dynamical, increasing complexity and evolution (change during execution) and distribution, logical and geographical, of IoT systems [14].

These solutions also possess several disadvantages for non-technical *end-users*. Consider a Node-RED system orchestrating a user's smart home with multiple devices. Even in situations where there are only a couple of rules defined, it can be challenging to understand why a specific event took place due to the overwhelming data flow that results from these. Furthermore, just a small amount of rules can already lead to a system not possible to visualize in a single screen [15]. The more rules one adds, the harder it becomes to grasp what the system can do conceptually. Part of the reason is that these solutions are built to be *imperative*, not *informative*; current solutions mostly lack in meta-facilities that enable the user or the system to *query* itself [16].

Several works highlight the issues that users have when configuring and understanding trigger-action programs [17, 18]. Huang and Cakmak in their work identify that ambiguities between *trigger types* (states and events) and *action types* (instantaneous, extended, and sustained actions), lead users to misconstrue and misinterpret their rules (the authors state that "people create different programs given the same prompt and are still in disagreement in their interpretations after having created programs themselves") [17]. Ghiani et al. mention similar issues in their work and emphasize that different individuals understand the same *concept* or *metaphor* differently, which also increase the proneness to errors and the difficulty to understand the programmed rules [18].

Some of our previous work attempt to enhance visual programming solutions, namely, Node-RED, with some additional features that attempt to ease the process of understanding, debugging and evolving IoT systems (*e.g.*, add a new sensor or service to an already existing system). Observability of the system was improved by adding visual inspection of the information which flows through the nodes, better system exploration was added by enhancing the debug capabilities through breakpoints and removing the need to re-deploy, and, lastly, runtime modification capabilities were added that allow the injection of messages during runtime. While, in overall, this approach optimized the development time and reduced the number of failed attempts to deploy the system, it does not address the issues with the misunderstanding of the metaphors used nor the ambiguity between *trigger types* and *action types* [19]. Similarly, other authors purpose these enhancements, namely, the support for debugging the trigger-action rules in visual solutions [20, 21].

Another common, and, sometimes, complementary, alternative to visual programming, is the use of conversational assistants (also known as voice assistants). There exist a plethora of conversational assistants in the market, such as Google Assistant, Alexa, Siri and Cortana (see [22] and [23] for a comparison of these tools) which are capable of answering natural language questions. Recently, these assistants have gained the ability to interact with IoT devices, with Ammari et al. identifying IoT as the third most common use case of voice

[1] Also known as trigger-action programming (TAP).

assistants [24].

Amongst the most common features they provide is allowing direct interaction with sensing and actuating devices, which enables the *end-user* to *talk* to their light bulbs, thermostats, sound systems, and even third-party services. The problem with these solutions is that they are mostly comprised of *simple* commands and queries directly to the smart devices (e.g., *is the baby monitor on?*", "*what is the temperature in the living room?*", or "*turn on the coffee machine*". These limitations mean that although these assistants do provide a comfortable *interaction* with devices, a considerable gap is easily observable regarding their capabilities on *managing* a system as a whole and allowing the definition of rules for how these *smart spaces* operate. Even simple rules like "*close the windows every day at 8 pm*" or "*turn on the porch light whenever it rains*" are currently not possible unless one manually defines every single one of them as a capability via a non-conversational mechanism. Furthermore, most assistants are deliberately locked to specific vendor devices, thus limiting the overall experience and integration.

One can conclude that although current smart assistants can be beneficial and comfortable to use, they do not yet have the complexity and completeness that other systems like Node-RED. Meanwhile, visual programming solutions are still far too technical for the common *end user*. In this paper, we propose a system that tackles the problem of *managing* IoT systems in a conversational approach, towards shortening the existing feature gap between assistants and visual programming. Parts of this work are summarized from Lago [25] master's thesis.

The rest of this document is structured as follows: Section 2 provides a summary of related works which identify open research challenges; in Section 3 we propose our approach to supporting *complex* queries in conversational assistants, which implementation details are further presented in Section 4. Section 5 presents the experimental setup and Section 6 presents the carried feasibility study to evaluate our approach using simulated scenarios and experimental studies. Finally, Section 7 delineates several research directions for the present work and in the scope of the state-of-the-art, and Section 8 drafts some closing remarks.

2. Related Work

There exists some work in this area that recognizes the problem of controlling and managing IoT infrastructures by an *end-user* via several approaches beyond trigger-action and other visual programming solutions. Within the scope of this work, this section presents only literature that focuses on works that integrated speech-based components within their solutions.

Kodali et al. [26] present a home automation system to *"increase the comfort and quality of life"*, by developing an Android app that can control and monitor home appliances using MQTT, Node-RED, IFTTT, Mongoose OS and Google Assistant. Their limitations lie in that the *flows* must have been created first in Node-RED, and the conversational interface is used to trigger them, ignoring all the *management* activities.

Austerjost et al. [27] recognized the usefulness of voice assistants in home automation and developed a system that targets laboratories. Possible applications reported in their paper include a stepwise reading of standard operating procedures and recipes, recitation of chemical substance or reaction parameters to control, and readout of laboratory devices and sensors. As with the other works presented, their voice user interface only allows controlling devices and reading out specific device data.

He et al. [28], concludes that, even with conversational assistants, most of IoT systems have usability issues when faced with complex situations. As an example, the complexity of managing devices schedules rises with the number of devices and the shared conflicting preferences of household members. Nonetheless, as concluded by Ammari et al. [24], controlling IoT devices is one of the most common uses of such assistants.

Agadakos et al. [29] focus on the challenge of understanding the causes and effects of an action to infer a potential sequence. Their work is based on a mapping the IoT system' devices and potential interactions, measuring expected behaviours with traffic analysis and side-channel information (*e.g.*, power) and detecting causality by matching the mapping with the collected operational data. This approach would potentially allow the *end user* to ask *why is something happening*, at the cost of modified hardware and a convoluted side-channel analysis. They did not attempt to port their findings into a conversational approach.

Braines et al. [30] present an approach based on Controlled Natural Language (CNL) — natural language using only a restricted set of grammar rules and vocabulary — to control a smart home. Their solution supports (1) *direct question/answer exchanges*, (2) *questions that require a rationale as response* such as *"Why is the room cold?"* and (3) *explicit requests to change a particular state*. The most novel part of their solution is in trying to answer *questions that require a rational response*; however, they depend on a pre-defined smart home model that maps all the possible causes to effects.

Kang et al. [31] explore the use of multi-modal interaction within IoT systems — combining voice and gesture interactions — as a way of addressing the scalability and expressiveness supported by existing IoT-vendors mobile applications and voice assistants. Although most of the participants who took part in the study responded positively to many interaction techniques, one of the identified pitfalls was the lack of robustness of the voice assistant that failed to understand the user commands.

Several other works [32, 33, 34] combine the use of voice assistants with IFTTT, using the later to define the system rules. While the primary control mechanism over the IoT system is voice-based, it is mostly used to trigger the IFTTT specified rules, depending on the rules' definition in a form-based visual interaction. Thus, it is also limited by them.

An empirical study by Ammari et al. [24] identifies IoT as one of the most common uses of voices assistants. In their study, users identified as the main drawbacks of the use of voice assistants the (1) lack of spatial and temporal contextualization and (2) lack of support for dynamic instructions (macros). Concerning (1) such awareness would allow the assistant to know where the user is physically at any point in time, thus acting in accordance (*e.g.*, turn on the lights in the room where the user is located without the need to provide further context). Regarding point (2), the users point to the need of creating macros to simplify their interactions with the devices (*e.g.*, supporting rules such as *when leaving home, turn off all the lights, close the garage door and reduce the thermostat temperature*).

To the best of our knowledge, no already-existent solution simultaneously provide: (1) a non-trivial management of an IoT system, (2) be comfortable and easy to use by a non-technical audience, and (3) allow the user to understand better how the system is functioning. By *non-trivial* we mean that it should be possible to define new rules and modify them via a conversational approach, achieving a *de facto* integration of multiple devices; not just directly interacting with its basic capabilities. The comfort would be for the user not to have to move or touch a device to get his tasks done (*i.e.*, using voice), or edit a Node-RED visual flow. As to understanding their system's functioning, we mean the ability to grasp *how* and *why* something is happening in their smart space. This last point, combined with the other two, would ideally allow someone to ask why something happens.

3. Solution Overview

We propose the development of a conversational assistant dedicated to the management of IoT systems that is capable of defining and managing complex system rules while providing information about the running system. Our prototype is called **Jarvis**, and is available as a reproducible package [35].

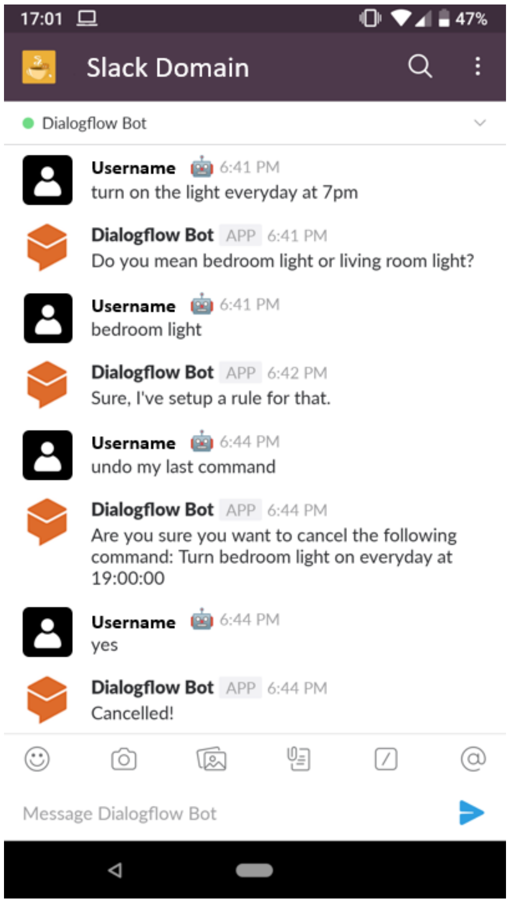

Figure 3: Chat with Jarvis by Slack integration.

An example interaction with **Jarvis** by text messages on Slack can be seen in Fig. 3. Jarvis provides users with several features with the aim of covering most of the interactions a user could have with physical smart spaces. The choice of this functionalities were based on the most common actions one can find in similar works and surveys [36], including those identified by [24] as main drawbacks in voice assistants. An empirical survey that attempts to systematize end-users actions can be found in [37], which gathered 177 smart home scenarios, further categorizing them into seven distinct sets.

Causality and rules queries and harder to find in the literature, as they represent the least explored areas. We have thus chosen to support the following functionalities:

Direct actions Single direct action that happens instantly, *e.g.*, "*Turn on the light*" or "*What is the current temperature of the kitchen?*";

Delayed actions Single delayed action that happens after a certain time period, *e.g.*, "*Turn on the light tomorrow at 5pm.*";

Repeating actions Defines a rule for an action that should be performed every day, *e.g.*, "*Turn on the light every day at 5 pm.*";

Event-triggered actions Creates an action that is performed upon a certain event, such as an activity of another device or a change of a device's status, *e.g.*, "*Turn on the light when the bedroom motion sensor is activated.*";

Causality queries Used when the user wants to know why a certain condition is true or why a certain action took place, *e.g.*, "*Why did the light turn on?*";

Alias actions Used for the user to create an action/event that associates to a custom phrase, *e.g.*, "*Make an alias for 'party time'.*" [system asks what to set the alias for] "*Turn on all lights.*";

Rules query Used to know which rules are defined for a device (allowing to change them), *e.g.*, "*What rules are defined for the living room light?*";

Cancel command Cancel the last user command. If that was a direct action command, the action is undone, and, if the command was a rule command, the rule is cancelled, *e.g.*, "*Cancel my last command.*".

Jarvis also uses *contextual awareness* in order to improve the user's experience and make the interaction resemble a real human-human interaction. *Contextual awareness* allows Jarvis to understand the meaning of a user query based on queries issued previously, which can be applied in many scenarios:

Device specification that is used when the device specified in an action query is unclear or ambiguous so that the user can specify the device he wants to choose.

User: "*Turn on the light.*"
Jarvis: "*Do you mean the living room light or the bedroom light?*"
User: "*The bedroom light.*"
Jarvis: "*Sure, light turned on.*"

Updating system rules which context is used to enable following-up of the *Rules query* action, allowing to change the presented rules.

User: "*What rules are defined for the bedroom light?*"
Jarvis: "*You told me to turn the bedroom light on everyday at 8 AM.*"
User: "*Okay, change it to 7:50 AM.*"
Jarvis: "*Sure, rule changed.*"

Causality queries which context enables the user to have a dialog to better grasp the reason why something happens (instead of a single direct answer which could be not understood by the user). **User**: "*Why did the toaster turn on?*"
Jarvis: "*You told me to turn it on at 10 AM.*"
User: "*Okay, change it to 9 AM.*"
Jarvis: "*Sure, toaster timer was changed.*"

It is noticeable that in all of the examples above, the second user query would be meaningless on its own. However, it makes sense when represented along with the previous user query and Jarvis' first answer. These examples show how *contextual awareness* can make interactions with Jarvis feel more natural, which improves the user's experience.

To ease the integration with nowadays systems and provide us with an *experimental reproducible environment*, we integrated the interface with some existing platforms, namely: Google Assistant [38] and Slack [39]. Integration with other services is also possible, and one can interact with Jarvis both via *voice* and *text*.

4. Implementation Details

Fig. 4 presents the high-level software components of Jarvis. Each component and corresponding techniques are explained in the following subsections.

4.1. Conversational Interface

To develop the conversational interface, we decided to opt for Dialogflow[2] as this platform provides built-in integration with multiple popular *frontends* and there exists extensive documentation for this purpose [40]. In

[2]Dialogflow, `https://dialogflow.com/`

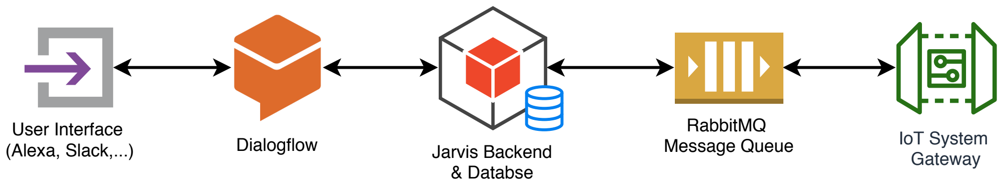

Figure 4: Jarvis overall architectural components.

this case, we used (1) the Slack team-communication tool (*cf.* Fig. 3), and (2) Google Assistant, so that both text and voice interfaces were covered. In the case of Google Assistant, the user may use any supported device paired with their account to communicate with Jarvis, following a known query prefix such as *"Hey Google, talk to Jarvis"*. Regardless of which type of interface is used, the result is converted to *strings* representing the exact user query and subsequently sent to Dialogflow's backend (thus overcoming potential challenges due to Speech Recognition), which are then analyzed using Natural Language Processing (NLP) techniques. Advancement of the existent NLP techniques made available by Dialogflow falls out-of-the-scope of this work.

4.2. Dialogflow Backend

Upon receiving a request, Dialogflow can either produce an automatic response or send the parsed request to a fulfilment *backend*. This component is thus responsible for parsing the incoming *strings* into a *machine understandable* format (JSON). There are a few key concepts that are leveraged in our implementation:

Entity. Things that exist in a specific IoT ecosystem can be represented by different literal strings; for example, an entity identified by `toggleable-device` may be represented by *"living room light"* or *"kitchen light"*. Additionally, entities may be represented by *other* entities. Dialogflow use of the `@` symbol (*i.e.* `@device`) for entities, and provides some system's defaults;

Intent. An intent represents certain type of user interaction. For instance, an intent named *Turn on/off device* may be represented by `turn the @device on` and `turn the @device off`. For a request such as *"turn the kitchen light on"*, Dialogflow understands that `@device` corresponds to *kitchen light* and provides that data to the fulfilment backend;

Context. Contexts allow intents to depend on previous requests, enabling the creation of context-aware interactions. These are what supports queries such as *"cancel that"* or *"change it to 8AM"*.

Multiple *intents*, *entities* and *contexts* were defined in Jarvis and the main ones are illustrated in Fig. 5. Here we provide in detail one of its *intents*:

Event Intent

Usage Creates an action that is performed upon a certain event, such as an activity of another device or a change of a device's status.

Definition `@action:action when @event:event`

Example *Turn the bedroom light on when the living room light turns off.*

With the above definitions, this component takes requests and builds the corresponding objects containing all actionable information to be sent to the Jarvis backend for further processing. For that, Dialogflow generates a JSON object that contains the exact user query, but also an identifier for the intent type, identifiers for the recognized entities, relevant contextual metadata and default answers (if any were specified in the Dialogflow configuration UI). This JSON is sent to the Jarvis backend via an HTTP request, to which Jarvis responds with a JSON containing the intended response along with other possible data such as contextual metadata.

4.3. Jarvis Backend

For each of the intents defined in Dialogflow, this component provides an equivalent class responsible for handling that intent, also named *handler classes*. Jarvis makes use of a MEDIATOR pattern to assign the handling of each user query to the right handler class.

Each *handler class* provides the same methods to the mediator, the main of each being a 'handle' method

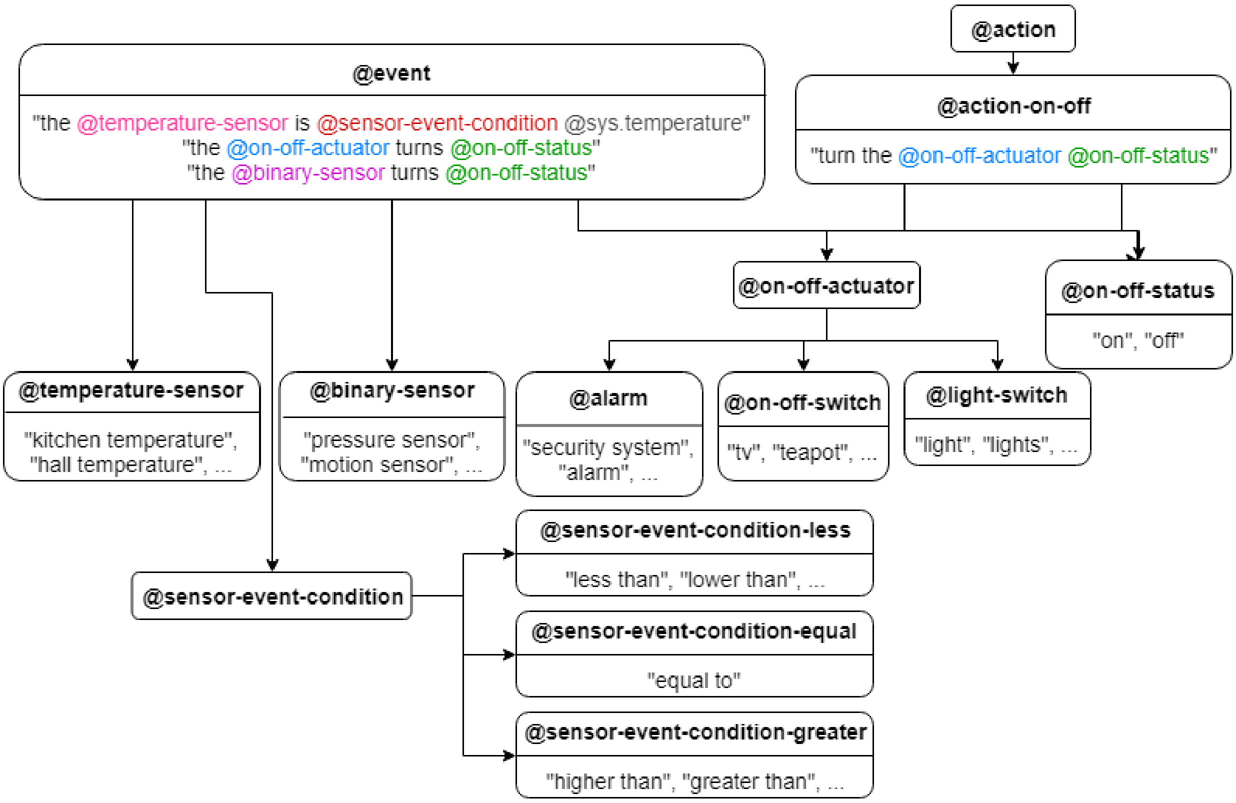

Figure 5: Main entities defined in Jarvis' Dialogflow project.

that takes in the user query as represented by Dialogflow's JSON object, returning the resulting JSON which should be sent to Dialogflow, containing Jarvis' response.

The *handler classes* are responsible for (a) parsing the request, (b) validating its request parameters (*e.g.* device name or desired action), and (c) generating an appropriate response. An overview is provided in Fig. 6. Should the request contain errors, an *explanatory* response is returned. When all the parameters are considered valid, but the intended device is *unclear* (*e.g.* user wants to turn on the light; however, there is more than one light that can be the target of the command), the generated response specifically asks the user for further clarification in order to gain *context*.

Additionally to Dialogflow's JSON representation of the user query, the Jarvis backend represents user commands using the COMMAND design pattern. This provides a straightforward way to *execute*, *cancel* and *undo* mechanisms, as well as keeping a history of performed actions, which proves especially useful for *causality queries*.

This internal representation of commands makes use of the Web Things API [3]. This API documents a symbolic representation of multiple devices along with their capabilities, which is useful for the Jarvis backend to be aware of a device's capabilities and features. This representation is what enables Jarvis to know whether a specific action (*e.g.* turning something on) applies to a particular device (*e.g.* a light).

4.3.1. Contextual awareness.

The first example of *contextual awareness* happens when the user makes a query with an unclear device. Here, Jarvis sets *contextual metadata* on the response set to Dialogflow. This metadata is then re-sent to Jarvis by Dialogflow on the following user query, which allows Jarvis to understand interactions such as:

User: "*Turn on the light.*"

Jarvis: "*Do you mean the bedroom light or the kitchen light?*"

User: "*The second one.*"

[3] Web Thing API, `https://webthings.io/`

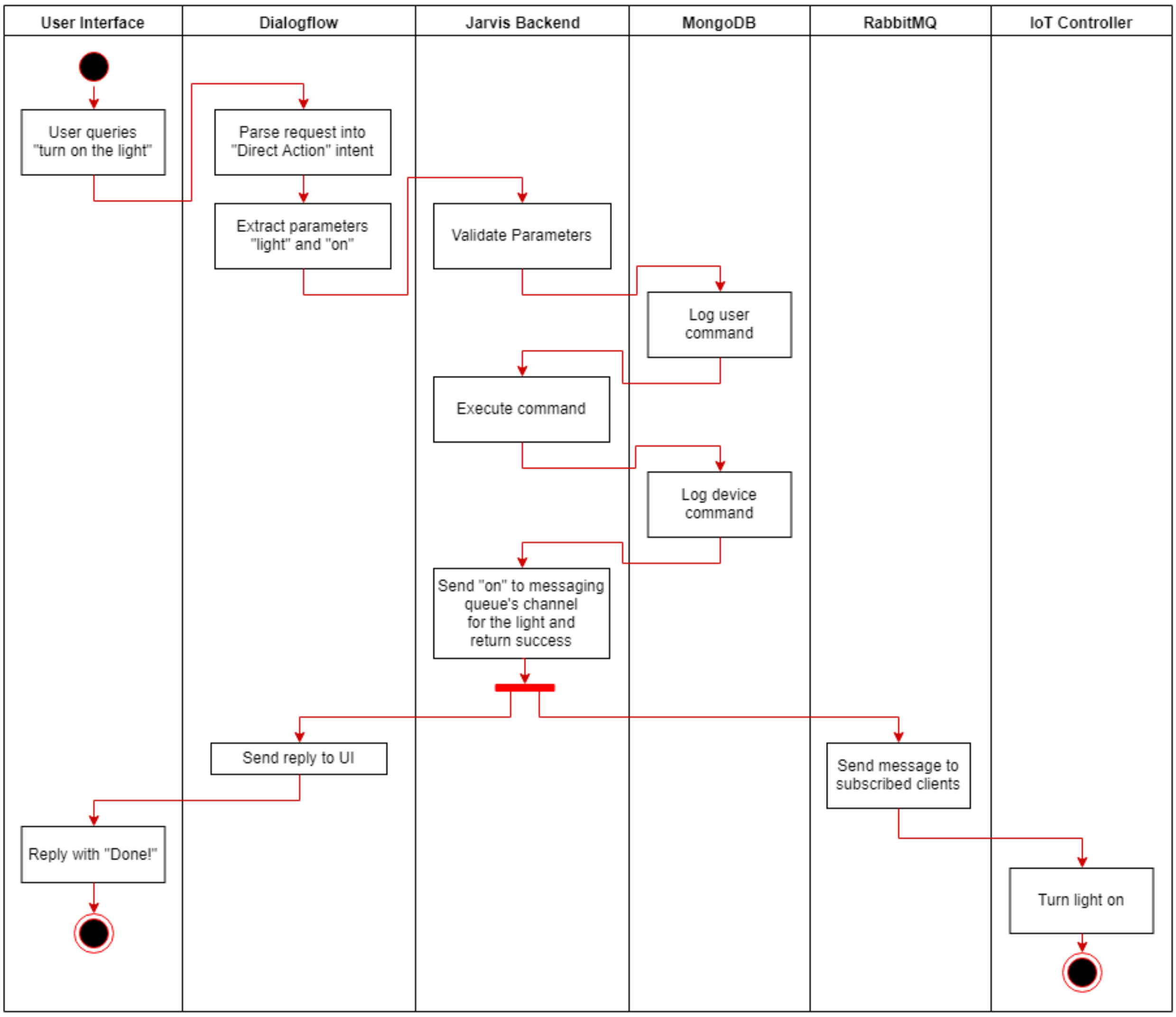

Figure 6: Sequence diagram for the parsing and execution of the query *turn on the light*.

Because of the *contextual metadata* set by Jarvis during the second response, when the user says "*The second one.*", Jarvis knows that the user is referring to the "*kitchen light*", and therefore knows that it must continue the initial query and turn on that device.

In the example above, the second user query is assigned by the *mediator* to a specific *handler class* which is able to decode the contextual metadata and generate the corresponding user *command*.

4.3.2. Period Actions.

For most intents, such as *direct actions* or *"why did something happen?"* queries, the effects are immediate. However, *period actions*, *events* and *causality queries* require a different design approach so that they can perform actions on the backend without the need for a request to trigger them.

A *period action* is an intent that must be carried and then undone after a certain period (*e.g.* *"turn on the light from 4 pm to 5 pm"*). In these scenarios, the Jarvis backend generates a *state machine* to differentiate between all the different action status, such as (a) nothing has executed yet (before 4 pm), (b) only the first action was executed (after 4 pm but before 5 pm), and (c) both have been executed (after 5 pm). We use a combination of *schedulers* and *threads* to guarantee proper action, and abstract all these details inside the COMMAND pattern. the same strategy applies for rules such as *"turn* *on the light every day at 5 pm"*, with the appropriate state machine and scheduler modifications.

In these examples, the already mentioned COMMAND representation becomes useful once again since it allows the system to manage these period actions easily. For instance, if the user wishes to change an active rule (*e.g.*, *"turn on the light from 4 pm to 6 pm"* instead of *"turn on the light from 4 pm to 5 pm"*), the Jarvis backend can cancel the active *command*, create a new instance with the updated rule and start it immediately. This update of an active *command* is itself represented as a command, which also allows the user to revert unintentional changes to other rules.

4.3.3. External Events.

This state-machine mechanism is different for actions that are the result of external events such as *"turn on the kitchen light when the presence sensor is activated"*. These are notably different because, although direct actions and period actions depend only on the internal state of the Jarvis backend, event-bound actions are dependant on analyzing external events such as a sensor changing its state.

To implement this functionality, we leverage a *publish-subscribe* approach which orchestrates multiple unique and identifiable *message queues*. Each message queue is associated with one or multiple devices, and it serves as a bidirectional communication layer between them and the Jarvis backend. For instance, when Jarvis wishes to change the state of a certain device, it publishes a message on the respective queue with a format that identifies the specific device to change and what that change requires. It is then the responsibility of that device's controller to read this message and perform the change. Messages published on these queues also leverage the Web Things API.

When it comes to events, communication happens in the reverse order. Each time a sensor's value changes (*e.g.*, a motion sensor is triggered or the temperature changes), that device's controller publishes a descriptive message on the message queue. The Jarvis backend then uses observers that read the message and decide whether any active COMMAND is responsible for handling it. If so, it calls a method on that command that handles the message.

This means that a user query such as *"turn on the kitchen light when the presence sensor is activated"* generates a COMMAND that knows it must handle changes to the presence sensor, such that when this happens, this command is called by the observer, causing the light to be changed accordingly.

4.3.4. Causality Queries.

These relate to the user asking why something happened (*e.g.*, *"why did the light turn on?"*). These are a unique feature of Jarvis which are very useful for users not only because they allow them to remember what are the operation rules of their system, but also because they allow users to easily change how their system works with nothing but their voice.

To implement them, we augment each COMMAND such that each command can determine whether it can cause a specific condition to be true. For instance, the command *"turn on the light when the presence sensor is activated"* knows that a possible consequence of its operation is the condition *"light is turned on"*.

With this augmentation, when the user queries Jarvis on why some condition happened, Jarvis can iterate through the log of recently executed commands and return the latest one that could have caused the queried condition, providing an informative answer (*e.g.*, *"because you asked me to turn it on at 3:56 pm"*).

However, there might exist multiple rules may have caused the condition to be true, in which case it is not enough to blame the latest logged command. In order to expand this functionality to provide more accurate answers, we considered three different approaches:

Return the immediate possible cause This is the currently implemented approach. It is likely to provide an accurate answer in the sense that the response is always the latest action that caused the queried event. Nevertheless, this does not necessarily imply that it is the most relevant cause (*e.g.*, if multiple commands could cause the queried condition, the first of these was the one that first led to that condition).

Return the first possible cause In some scenarios, multiple rules might have been involved in the change of the current system state, and they might either be part of a "causal chain", or simply overlap in their outcome. It is debatable whether the most relevant action in the chain would be the most immediate, the root event, or anything in between. However, in the case of overlapping, it seems that the first event to have occurred (in the sense of sequence) might be the most reasonable to blame — since it is the one that transited the state — and which was latter "reinforced" by other causes (*e.g.*, if multiple rules could have caused the light to turn on, only the first of which caused the light's state to be changed). Hence, this first rule might be the most relevant answer in some cases.

Use relevance heuristic A relevance heuristic could provide the benefits of both of the previous approaches, perhaps being even better. In a situation where multiple rules or events could have caused the queried condition, using a heuristic could provide an answer that was more useful to the user. For instance, if both a period event and an event action could have caused the condition, a heuristic could consider the event to be a more relevant condition since it is caused by external interactions rather than the well-defined mechanisms defined by the user.

Another non-trivial scenario is where the explanation is due to a chain of interconnected rules. Here, it seems that one can (a) reply with the complete chain of events, (b) reply with the latest possible cause, or (c) engage in a *conversation* through which the user can explore the full chain of events as they deem adequate (e.g., *"tell me more about things that are triggered by rain"*). In this work, we opted to use the earliest possible cause for the first scenario, and the latest for the second; more complex alternatives can be found in [30, 29].

4.4. Interaction with IoT devices

For the interaction with the physical IoT, we chose a simple yet functional set of technologies that would allow us to validate the functionality of the Jarvis backend. We used RabbitMQ [41] as the message queue system, since it supports a variety of protocols (such as AMQP, STOMP and MQTT), allowing easy communication with devices through simple path strings (*e.g.*, `/house/kitchen`). The message queue system allowed Jarvis backend to communicate with the IoT devices while being agnostic of their physical location on the network. An alternative setup could require the backend to know the *IPs* of each individual device, which would require much more maintenance if those addresses changed over time.

In order for Jarvis to know which devices exist in the system, how to communicate with them and what capabilities they have, a Device Registry [42] was set up, and such information was stored using a MongoDB [43] document-based database. This database was also used to store the history of user queries and executed commands, which allows the system to provide features such as the causality queries even if it is temporarily shut down.

The direct interaction with the IoT devices was simulated using *Python* scripts that publish the changes in states of *IoT* devices on the message queues, as well as read instructions provided by Jarvis and apply them to the respective devices.

In the experimental setup we used in the validation of this project, the Jarvis was deployed in a virtual private server (VPS) such that it could easily be accessed from any location.

5. Experimental Setup

To understand how Jarvis compares to other systems, we established a baseline based on (1) a visual programming language, and (2) a conversational interface. Node-RED was picked amongst the available visual programming solution, as it is one of the most popular visual programming solutions [44]. It follows a flow-based programming paradigm, providing its users with a web-based application through which they can manage rules via *connections* between *nodes* that represent devices, events and actions [12]. Google Assistant was selected for the conversational interface due to its naturality[4]. There are plenty of ways users can interact with it: (a) the standalone Google apps, (b) built-in integration with Android and Chrome OS, or (c) with standalone hardware such as the Google Home. We compare to this baseline according to two criteria: (1) the *number of different features*, and (2) their *user experience* in terms of easiness of usage and intuitiveness. For the first, we created a list of simulated scenarios to assess the ability to manage IoT systems. We then performed a feasibility experiment with users to assess the second criteria.

5.1. Simulated Scenarios

A total of 10 simulated tasks was performed with the goal of comparing Jarvis with two solutions available in the market: Node-RED and Google Assistant. Table 1 summarizes the comparison of our prototype to the chosen baseline.

The (1) *one-time action* refers to a direct trigger of a device, which is possible in both voice assistants and through the Node-RED interface. The (2) *one-time action with unclear device* refers to actions like *"turn on the light"* with which Jarvis asks the user to clarify which device he means based through responses such as *"do you mean the bedroom or living room light?"*. Queries such as (3) *delayed action*, (4) *period action*, (5) *daily repeating action* and (6) *daily repeating period*

[4]The work by López et al. [23] compares Alexa, Google Assistant, Siri and others, and claim that although "*Siri was the most correct device (...) Google assistant was the one with the most natural responses"*.

Table 1: Simulated scenarios comparison.

ID	Scenario	Jarvis	Google Assistant	Node-RED
1	One-time action	●	●	●
2	One-time action w/unclear device	●	·	·
3	Delayed action	●	·	●
4	Periodic action	●	·	●
5	Daily repeating action	●	·	●
6	Daily repeating period action	●	·	●
7	Cancel the last command	●	·	·
8	Dynamic creation of event rules	●	·	·
9	Rules defined for device	●	·	·
10	Causality query	●	·	·

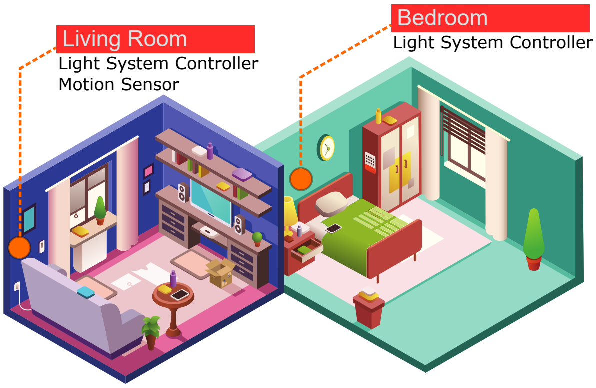

Figure 7: Visualization of the scenarios used for the feasibility experiment.

action are possible to carry using the Jarvis assistant and with the Node-RED solution. The query (7) *cancel the last command* refers to the ability to undo the last action or rule creation by explicitly saying that, and while that is possible to be carried on Jarvis, neither Google Assistant nor Node-RED support this behaviour.

In the case of an (8) *event rule*, the system must support the dynamical creation of trigger-action rules based on an event (*e.g.*, the trigger of a motion sensor or when a button is clicked), which is possible using Jarvis, but in Node-RED requires manual changes to the programmed flows. Query (9) *rules defined for device* refers to the user performing queries that require introspection, such as *"what rules are defined for the bedroom light?"*, which Jarvis is capable of, but this capability is not available in Google Assistant. In Node-RED this can be accomplished up to a certain point by visual inspection of the flows, though it has several limitations[5]. Concerning (10) *causality query*, the solution should provide a reasonable cause for a given event, which is only possible in Jarvis.

It is observable that our prototype provides several features that are not present in either the Google Assistant or Node-RED. Both of these products do a lot more than these features. However, in regards to managing smart systems, the advantage of Jarvis is evident, especially when compared to the Google Assistant given that the only type of feature it supports are *one-time direct actions* [24]. Our second conclusion is that it is possible to bring some of the features currently available in visual programming environments to a conversational interface; the converse (how to bring conversational features to Node-RED), eludes the authors.

It is essential to mention that both Node-RED and the Google Assistant are systems with broader goals than just automating the management of IoT systems. Node-RED is capable of managing complex rules that connect multiple different systems. For instance, it allows users to send an automated email any time a tweet with a certain *hashtag* is published. The Google Assistant is also capable of many other features, such as listening to music or telling users about their upcoming flight reservations. Jarvis does not aim to provide any of these features, being tailored to IoT scope.

The comparison between these services and Jarvis on the limited scope of managing an IoT smart space is meant as a reinforcement of the value added by Jarvis in this limited scope, rather than downplaying the overall value and potential of the two systems used as comparisons.

6. Feasibility Experiment

In order to gain insight into how *end users* responded to a conversational approach, we performed a feasibility experiment with 17 participants. Our sample includes 14 participants without formal technological skills, with ages ranging from 18 to 51. The remained 3 participants were students enrolled in the Masters in Informatics Engineering. We made sure that (a) all participants were familiar with the necessary technologies, such as basic usage of smartphones and the Internet, and (b) that even non-native English participants had adequate speaking and understanding skills, given that the prototype of Jarvis was implemented in the English language.

[5]As an example of such limitation is that if more than one device is connected to the same message queue it can be very difficult to understand which device produced a particular outcome and thus hard to understand if a rule was trigger due to a specific device *event*.

6.1. Methodology

Each participant was given 5 tasks to be completed using the same scenario with the help of Jarvis, using Google Assistant as the system interface. The scenario consisted of a *smart home* with a *living room light*, a *bedroom light* and a *living room motion sensor*, as depicted in Fig. 7:

Task 0 (control) (T0) *Turn on the living room light*;

Task 1 (T1) *Turn the living room light on in 5 minutes*;

Task 2 (T2) *Turn the living room light on when the motion sensor triggers*;

Task 3 (T3) *Check the current rules defined for the bedroom light, and then make it turn on everyday at 10pm*;

Task 4 (T4) *Find out the reason why the bedroom light turned on. Ask Jarvis why it happened and decide whether the answer was explanatory.*

The only instructions given to participants were that they should talk to the assistant (using the mobile phone version) in a way that feels the most natural to them to complete the task at hand. Besides the tasks, participants were also given the list of IoT devices available in the simulated smart house that they would be attempting to manage through.

6.2. Variable Identification

For each of the tasks, we collected (1) whether the participant was able to complete it, (2) the time to complete, and (3) the number of unsuccessful queries. This count was made separately for (a) queries that were not understood by the assistant's speech recognition capabilities (*e.g.* microphone malfunction, background noise), (b) queries where the user missed the intention or made a syntactic/semantic error (e.g., *"turn up the lighting"*), and (c) valid queries that a human could interpret, but that Jarvis was unable to.

6.3. Subjective Perception

After completing the tasks, we introduced a non-conversational alternative (Node-RED), explaining how all tasks could have been performed using that tool. We inquired the participants whether they perceived any advantages of Jarvis over such a tool and whether they would prefer Jarvis over non-conversational tools. Finally, the participants were asked if they had any suggestions to improve Jarvis and the way it handles system management.

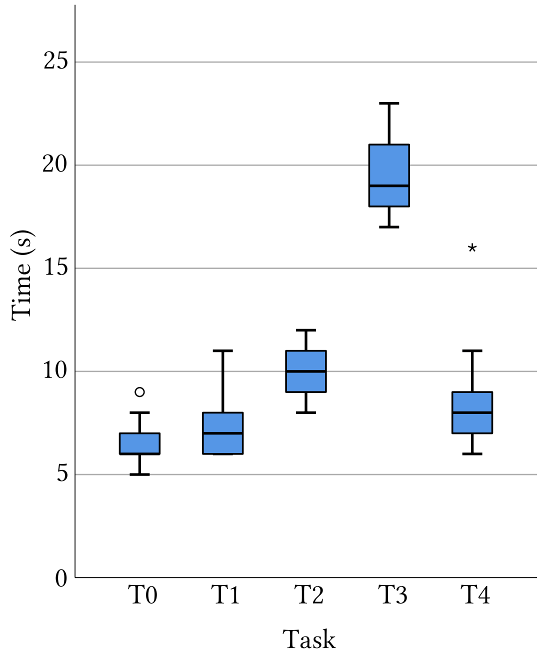

Figure 8: Boxplot of task completion time (s) per task.

6.4. Results

Table 2 compiles the results observed during the study, each row representing a task given to the participant. Each column means:

Task Identification of the task (T0—T4);

Done Percentage of participants that completed the task successfully;

Time Time in seconds that participants took to complete the task;

IQ (G.A.) Number of occurrences of queries that were incorrect due to the Google Assistant (G.A.) not properly recognizing the user's speech;

IQ (User) Number of occurrences of queries that were incorrect due to the user not speaking a valid query;

IQ (Jarvis) Number of occurrences of queries that were incorrect due to Jarvis not recognizing a valid query;

IQ (Total) Total count of invalid queries, *i.e.* sum of *IQ (G.A.)*, *IQ (User)* and *IQ (Jarvis)*.

Table 2: Experimental results (task completion rate, task time and number of incorrect queries), including average and standard deviation.

		Time (s)		# IQ (G.A.)		# IQ (User)		# IQ (Jarvis)		# IQ (Total)	
Task	Done (%)	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
T0	94%	6.41	1.12	0.24	0.56	0.12	0.33	0.24	0.56	0.59	0.87
T1	94%	7.35	1.46	0.24	0.44	0.25	0.50	0.24	0.56	0.53	0.72
T2	88%	9.94	1.20	0.35	0.70	0.35	0.61	0.53	0.80	1.24	1.15
T3	100%	19.71	1.96	0.24	0.56	0.24	0.44	0.47	0.62	0.94	0.83
T4	94%	8.65	2.32	0.29	0.47	0.29	0.59	0.12	0.33	0.71	0.85

6.5. Discussion

The complexity of the queries increases from **T0** to **T3** since the queries require more words or interactions. This is reflected by the corresponding increase in task completion time, as seen in Fig. 8. The values related to incorrect queries show some occurrences at the (voice) assistant level, which means the speech recognition failed to translate what the participants said correctly. Although this does not have implications on the evaluation of Jarvis, it does indicate that this sort of systems might be harder to use due if they are not multilingual.

Directly comparing the time needed to complete a task to what would be needed to perform it in a visual programming solution such as Node-RED is meaningless; either the task is not defined, and that would require orders of magnitude longer than what we observe here, or the task is defined and the times will be obviously similar. Similarly, we also observe a few instances of incorrect queries due to grammar mistakes or semantically meaningless, *cf. IQ (User)*, and therefore did not match the sample queries defined in Dialogflow. Nevertheless, there where grammatically incorrect user queries such as *"turn on lights"* but which still carries enough information to understand what the user's intent is.

We consider as a more serious issue the number of *valid* sentences that were considered incorrect queries by Jarvis, *cf. IQ (Jarvis)*, as it can be seen in Fig. 9. These could have been caused by either a mispronunciation of a device's name or a sentence structure that is unrecognizable by the Dialogflow configuration. This possibly represents the most severe threat to our proposal, to which we will later dedicate some thoughts on how to mitigate it. Nonetheless, the success rate of all tasks is very high (always higher than 88%), which provides evidence that the system might be intuitive enough to be used without previous instruction or formation. These points were reflected by the participants' subjective perception, where they claimed Jarvis to be easy to use, intuitive, and comfortable; ultimately, these would be the deciding factors for end-users to prefer Jarvis over a non-conversational interface.

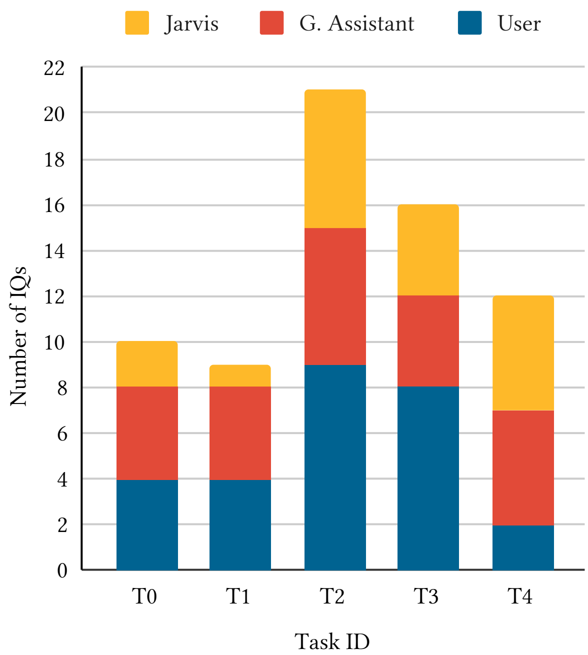

Figure 9: Bar chart of the number of IQs per task per component.

An additional observation was stated by some users pertaining Jarvis' answers, particularly those regarding causality queries (**T4**), where they claimed that if the provided response were too long, it would become harder to understand it due to the sheer increase of conveyed information. A possible solution for this problem would be to use a hybrid interface that provides both visual and audio interactions. However, there could be other approaches, such as an interactive dialogue that shortens the sentences.

In terms of subjective perception, when participants were inquired about their preference on visual programming solutions and the used voice interface, Jarvis, all of them pointed to conversational assistants as their preference, mostly due to its "ease of use", "commodity" and "accessibility". The most often referred downside were the issues with voice recognition ("margin of error that comes with voice recognition"). The participants mentioned that the main drawback of visual programming tools is the need to understand more technicalities on how the devices communicate and which actions (sensing/actuating) they can perform ("knowledge of how the hardware works"), and referred as the main advantage the large number of integrations that visual tools typically provide which lack in most conversational ones.

6.6. Threats to Validity

Empirical methods seem to be one of the most appropriate techniques for assessing our approach (as it involves the analysis of human-computer interaction), but it is not without liabilities that might limit the extent to which we can assess our goals. We identify the following threats:

Natural Language Capabilities where queries like *"enable the lights"* might not be very common or semantically correct, but it still carries enough information so that a human would understand its intention. The same happens with device identification, such as when the user says *turn on the bedroom lights*, and the query fails due to the usage of the plural form. During our study, we observed many different valid queries that did not work due to them not being covered by the Dialogflow configuration. This can be further addressed by creating a more extensive list of entities[6], and by training the DialogFlow model with more combinations of those entities;

Coverage error which refers to the mismatch between the *target* population and the *frame* population. In this scenario, our target population was (non-technical) end-users, while the frame population were all users who volunteered to participate;

Sampling errors are also possible, given that our sample is a small subset of the target population. Repeating the experience would necessarily cover a different sample population, and likely attain different results.

[6]The basic definition of an entity is that of a list of possible values, and thus, for more coverage, it should contain several different ways in which certain words can be expressed.

We attempt to mitigate these threats by providing a reproducible package [35], which allows this work to be easily reproduced and validated by other researchers with a minimal setup. Apart from the configuration of the *Dialogflow* system, the rest of the Jarvis solution can be used via the published reproducible package.

7. Research Directions

Although the number of functionalities that Jarvis provides and given the feasibility of such an approach for IoT configuration and management, we identify the following research directions that would improve the solution (or any similar approach):

Engaging in longer but fragmented conversations that would allow users to digest information at their own pace. This could be particularly useful when providing causality explanations since the user could iteratively explore more about the queried cause only if they wish to do so;

Support competing interactions as these can create contradictions and/or repetitions in the system. As the smart home system increases in complexity, originating by the increase of connected and interacting IoT devices (human-to-device and device-to-device) and the number of interacting people within the household, it becomes harder to avoid and mitigate overlapping rules or competing interactions. Adding specific capabilities to deal with more complex scenarios with multiple users and multiple interacting devices might reduce the complexity of dealing with such scenarios;

Support for priorities and roles as the number of individuals and parties that interact with the system increases, overriding rules can be introduced that might lead to both unintended consequences, as well as pose security and/or safety risk. Researching on how the system can identify which type of actions an user can request, as well as distinguishing between those that in tandem might lead to unforeseen consequences does not seem trivial;

Exploring different causality-finding algorithms as these might provide more insightful answers. As presented, the current prototype always determines as the cause of an event the latest possible action that could have caused it; however, the authors believe that exploring alternatives such as heuristics that change the approach depending on the type of logged events might provide more useful answers to users;

Understanding implicit causality relations between different events. For instance, if there is a light sensor close to a light, Jarvis turning on that light could trigger a change on that sensor, which the current prototype of Jarvis would not understand as correlated events. If Jarvis were to have a more *"semantic"* understanding of the system, it could perceive events like these as being related, which could further improve its answers to causality queries;

Supporting addition or removal of devices to the system. Jarvis currently uses an already configured database of devices to understand the system it is managing. Adding the capability to add or remove devices to the system would make Jarvis even more useful, particularly in a scenario where it would be used by end-users in their own spaces.

Supporting boolean operators in user queries. For example, when defining event rules, it would be useful to use multiple conditions with boolean ("and"/"or") operators. An example of this feature would be the query "Turn on the bedroom light if the motion sensor is activated and it is after 9 pm", where both conditions would have to be true in order to the action to be executed;

Privacy assurance most solutions, including Jarvis itself, depend on cloud-based NLP solutions to understand the user intents, which raises several concerns such as if the devices are always on (always listening), what is the history stored by the service providers (conversational logs) and how the data is managed (*e.g.*, third-party access) [24].

Being IoT one of the most common targets of conversational assistants *commands*, it becomes crucial to improve the user interaction with the devices by voice, mostly because existent solutions are limited, with the most only supporting *direct actions* [24].

8. Conclusions

In this paper we presented a conversational interface prototype able to carry several different management tasks currently not supported by voice assistants, with capabilities that include: (1) Delayed, periodic and repeating actions, enabling users to perform queries such as *"turn on the light in 5 minutes"* and *"turn on the light every day at 8 am"*; (2) The usage of contextual awareness for more natural conversations, allowing interactions that last for multiple sentences and provide a more intuitive conversation, *e.g.*, *"what rules do I have defined for the living room light?"*; (3) Event management that allows orchestration of multiples devices that might not necessarily know that each other exists, *e.g.*, *"turn on the light when the motion sensor is activated"*; and (4) Causality queries, to better understand how the current system operates, *e.g.*, *"why did the light turn on?"*.

Causality queries, specifically, are of great relevance, given that they are not supported by either conversational or visual tools. These queries provide an advance in the level of the conversational engagement with automated systems, therefore facilitating the management of smart spaces.

We conducted feasibility experiments with participants that were asked to perform specific tasks with our system. The overall high success rate shows the feasibility of our approach since the solution is intuitive enough to be used by people without significant technological knowledge. It also shows that most challenges lie in the natural language capabilities of the system, as it is hard to predict for any user queries that have the same intrinsic meaning. We thus conclude that incorporating recent *NLP* advances (that were beyond the scope of this work) would have a high impact in terms of making the system more flexible to the many different ways (correct or incorrect) that users articulate the same intentions.

Some of these improvements could even be easily made by implementing adjustments to the configuration of the Dialogflow tool. As mentioned, *user intents* are defined in the tool via sample queries. Therefore, merely diversifying the set of sample queries for each user intent, which could already be done by analyzing the incorrect queries from our controlled experiments, could provide significant improvements to the system.

All the experiment participants were using Jarvis for the first time when we ran the experiment. As happens with many other kinds of products, each user's experience could benefit from them getting to know the tool and getting more familiar with its features and capabilities. In other words, it is possible that repeated use of Jarvis would increase the user's familiarity and therefore reduce the occurrence of incorrect queries even further.

Nonetheless, by making a feature comparison, we can observe that Jarvis can implement many features that current conversational assistants lack, while simultaneously being more user-friendly than the available alternatives to IoT management (such as visual programming approaches). In overall Jarvis, or similar solution can ease and assist the process of configuring and managing IoT systems, significantly when the system in question increases in complexity, hindering the capability of end-users of understanding what is happening or which event lead to a specific outcome (and, possibly, correct the behaviour). As more than one person in a typical household might use these systems, it becomes useful to understand behaviours that perhaps were defined by other members and to edit defined behaviours on-the-fly without needing to re-program the system traditionally.

Although our work is mainly focused on smart-homes, the usage of IoT devices in industrial and other professional settings, such as health and bio laboratories, are also becoming increasingly common. In environments where bio-safety is paramount and touching devices might pose a risk, we see the technology here presented as having massive potential for traction and become virtual assistants to lab workers, helping in their routine tasks and even providing information and insights into their procedures.

Conflict of Interest

No conflict of interest to be declared.

Acknowledgement

This work was partially funded by the Integrated Masters in Informatics and Computing Engineering of the Faculty of Engineering, University of Porto (FEUP) and by the Portuguese Foundation for Science and Technology (FCT), under research grant SFRH/BD/144612/2019.

References

[1] A. S. Lago, J. P. Dias, H. S. Ferreira, Conversational interface for managing non-trivial internet-of-things systems, in: V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, J. Teixeira (Eds.), Computational Science – ICCS 2020, Springer International Publishing, Cham, 2020, pp. 384–397.

[2] F. Xia, L. T. Yang, L. Wang, A. Vinel, Internet of Things, International Journal of Communication Systems 25 (2012).

[3] B. Alghamdi, H. Fouchal, A mobile wireless body area network platform, Journal of Computational Science 5 (2014).

[4] I. Altintas, Building cyberinfrastructure for translational impact: The wifire example, Journal of Computational Science (2020).

[5] K. Qian, C. Claudel, Real-time mobile sensor management framework for city-scale environmental monitoring, Journal of Computational Science 45 (2020) 101205.

[6] J. Miranda, N. Mäkitalo, J. Garcia-Alonso, J. Berrocal, T. Mikkonen, C. Canal, J. M. Murillo, From the Internet of Things to the Internet of People, IEEE Internet Computing 19 (2015) 40–47.

[7] L. Mainetti, V. Mighali, L. Patrono, An iot-based user-centric ecosystem for heterogeneous smart home environments, in: 2015 IEEE International Conference on Communications (ICC), 2015, pp. 704–709.

[8] A. Zarzycki, Strategies for the integration of smart technologies into buildings and construction assemblies, in: Proceedings of eCAADe 2018 Conference, 2018, pp. 631–640.

[9] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, N. Mehandjiev, Meta-design: a manifesto for end-user development, Communications of the ACM 47 (2004) 33–37.

[10] A. Rahmati, E. Fernandes, J. Jung, A. Prakash, Ifttt vs. zapier: A comparative study of trigger-action programming frameworks, ArXiv abs/1709.02788 (2017).

[11] R. Gennari, L. U. Bozen-bolzano, A. Melonio, L. U. Bozen-bolzano, End-User Development, June, Springer, 2017. doi:10.1007/978-3-319-58735-6.

[12] P. P. Ray, A Survey on Visual Programming Languages in Internet of Things, Scientific Programming 2017 (2017) 1–6. doi:10.1155/2017/1231430.

[13] C. Prehofer, L. Chiarabini, From IoT Mashups to Model-based IoT, W3C Workshop on the Web of Things (2013).

[14] R. Seiger, C. Keller, F. Niebling, T. Schlegel, Modelling complex and flexible processes for smart cyber-physical environments, Journal of Computational Science 10 (2014).

[15] P. Janssen, H. Erhan, K. W. Chen, Visual dataflow modelling - some thoughts on complexity, in: Proceedings of the 32nd eCAADe Conference, 2014, pp. 547–556.

[16] J. P. Dias, J. P. Faria, H. S. Ferreira, A reactive and model-based approach for developing internet-of-things systems, in: 2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC), 2018, pp. 276–281.

[17] J. Huang, M. Cakmak, Supporting mental model accuracy in trigger-Action programming, UbiComp 2015 - Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (2015) 215–225. doi:10.1145/2750858.2805830.

[18] G. Ghiani, M. Manca, F. Paternò, C. Santoro, Personalization of Context-Dependent Applications Through Trigger-Action Rules, ACM Transactions on Computer-Human Interaction 24 (2017) 1–33. doi:10.1145/3057861.

[19] D. Torres, J. P. Dias, A. Restivo, H. S. Ferreira, Real-time feedback in node-red for iot development: An empirical study, in: 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2020, pp. 1–8.

[20] M. Manca, Fabio, Paternò, C. Santoro, L. Corcella, Supporting end-user debugging of trigger-action rules for iot applications, International Journal of Human-Computer Studies 123 (2019) 56 – 69.

[21] F. Corno, L. De Russis, A. Monge Roffarello, Empowering end users in debugging trigger-action rules, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, 2019, pp. 1–13.

[22] M. Mitrevski, Conversational interface challenges, in: Developing Conversational Interfaces for iOS, Springer, 2018, pp. 217–228.

[23] G. López, L. Quesada, L. A. Guerrero, Alexa vs. siri vs. cortana vs. google assistant: A comparison of speech-based natural user interfaces, in: I. L. Nunes (Ed.), Advances in Human Factors and Systems Interaction, Springer International Publishing, Cham, 2018, pp. 241–250.

[24] T. Ammari, J. Kaye, J. Y. Tsai, F. Bentley, Music, search, and iot: How people (really) use voice assistants, ACM Trans. Comput.-Hum. Interact. 26 (2019). doi:10.1145/3311956.

[25] A. S. Lago, Exploring Complex Event Management in Smart-Spaces through a Conversation-Based Approach, Master's the-

sis, Faculty of Engineering, University of Porto, 2018.

[26] R. Kishore Kodali, S. C. Rajanarayanan, L. Boppana, S. Sharma, A. Kumar, Low cost smart home automation system using smart phone, in: 2019 IEEE R10 Humanitarian Technology Conference (R10-HTC)(47129), 2019, pp. 120–125.

[27] J. Austerjost, M. Porr, N. Riedel, D. Geier, T. Becker, T. Scheper, D. Marquard, P. Lindner, S. Beutel, Introducing a virtual assistant to the lab: A voice user interface for the intuitive control of laboratory instruments, SLAS TECHNOLOGY: Translating Life Sciences Innovation 23 (2018) 476–482.

[28] W. He, J. Martinez, R. Padhi, L. Zhang, B. Ur, When smart devices are stupid: Negative experiences using home smart devices, in: 2019 IEEE Security and Privacy Workshops (SPW), 2019, pp. 150–155.

[29] I. Agadakos, G. Ciocarlie, B. Copos, T. Lepoint, U. Lindqvist, M. Locasto, Butterfly effect: Causality from chaos in the iot, in: International Workshop on Security and Privacy for the Internet-of-Things, 2018, pp. 26–30.

[30] D. Braines, N. O'Leary, A. Thomas, D. Harborne, A. D. Preece, W. M. Webberley, Conversational homes: a uniform natural language approach for collaboration among humans and devices, International Journal on Advances in Intelligent Systems 10 (2017) 223–237.

[31] R. Kang, A. Guo, G. Laput, Y. Li, X. A. Chen, Minuet: Multimodal interaction with an internet of things, in: Symposium on Spatial User Interaction, SUI '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 1–10. URL: https://doi.org/10.1145/3357251.3357581. doi:10.1145/3357251.3357581.

[32] D. S. K. Bheesetti, V. N. Bhogadi, S. K. Kintali, M. Zia Ur Rahman, A complete home automation strategy using internet of things, in: A. Kumar, S. Mozar (Eds.), ICCCE 2020, Springer Singapore, Singapore, 2021, pp. 363–373.

[33] T. Kim, Short research on voice control system based on artificial intelligence assistant, in: 2020 International Conference on Electronics, Information, and Communication (ICEIC), 2020, pp. 1–2. doi:10.1109/ICEIC49074.2020.9051160.

[34] M. Karthikeyan, T. S. Subashini, M. S. Prashanth, Implementation of home automation using voice commands, in: K. S. Raju, R. Senkerik, S. P. Lanka, V. Rajagopal (Eds.), Data Engineering and Communication Technology, Springer Singapore, Singapore, 2020, pp. 155–162.

[35] A. Lago, andrelago13/jarvis: Initial release, 2020. doi:10.5281/zenodo.3741953.

[36] J. Brich, M. Walch, M. Rietzler, M. Weber, F. Schaub, Exploring end user programming needs in home automation, ACM Trans. Comput.-Hum. Interact. 24 (2017). URL: https://doi.org/10.1145/3057858. doi:10.1145/3057858.

[37] D. A. Soares, Model-to-Model Mapping of Semi-Structured Specifications to Visual Programming Languages, Master's thesis, Faculty of Engineering, University of Porto, 2020.

[38] Google, LLC, Google assistant, your own personal google, 2020. URL: https://assistant.google.com/.

[39] Slack Technologies, Inc., Slack: Where work happens, 2020. URL: https://slack.com/.

[40] S. Janarthanam, Hands-on chatbots and conversational UI development: Build chatbots and voice user interfaces with Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio, and Alexa Skills, Packt Publishing Ltd, 2017.

[41] VMware, Inc, Rabbitmq, 2020. URL: https://www.rabbitmq.com/.

[42] A. Ramadas, G. Domingues, J. P. Dias, A. Aguiar, H. S. Ferreira, Patterns for things that fail, in: Proceedings of the 24th Conference on Pattern Languages of Programs, PLoP '17, The Hillside Group, USA, 2017, pp. 1–10.

[43] MongoDB, Inc., Mongodb, 2020. URL: https://www.mongodb.com/.

[44] N. K. Giang, R. Lea, M. Blackstock, V. C. Leung, Fog at the edge: Experiences building an edge computing platform, in: 2018 IEEE International Conference on Edge Computing (EDGE), IEEE, 2018, pp. 9–16.