LYSIS CHATBOT: A VIRTUAL ASSISTANT FOR IOT PLATFORMS

Raimondo Cossu[1], Roberto Girau[2], Luigi Atzori[3]

[1,3]Dept. of Electrical and Electronic Engineering, University of Cagliari and National Telecommunication Inter-University Consortium (CNIT), Research Unit of Cagliari, Italy., [2]Dept. of Computer Science and Engineering, University of Bologna, Italy

NOTE: Corresponding author: Roberto Girau, roberto.girau@unibo.it

Abstract – *The configuration and management of devices and applications in Internet of Things (IoT) platforms may be very complicated for a user, which may limit the usage of relevant functionalities and which does not allow its full potential to be exploited. To address this issue, in this paper we present a new chatbot which is intended to assist the user in interacting with an IoT platform and allow them to use and exploit its full potential. The requirements for a user-centric design of the chatbot are first analyzed, then a proper solution is designed which exploits a serverless approach and makes extensive use of Artificial Intelligence (AI) tools. The developed chatbot is integrated with Telegram to message between the user and the Lysis IoT platform. The performance of the developed chatbot is analyzed to assess its effectiveness when accessing the platform, set the main devices' parameters and request data of interest.*

Keywords – Chatbot, IoT platform, Lysis IoT, user experience

1. INTRODUCTION

In recent years, the development and deployment of chatbots to be used in several scenarios have risen significantly, and many businesses have opted to use these in their services. In many sectors, such as e-commerce, insurance, banking, healthcare, finance, legal, and others, chatbots are currently used to support the execution of a variety of business activities. Gartner Summits [1] predicts that over 70 % of customer interactions will involve emerging technologies such as Machine Learning (ML) applications, chatbots and mobile messaging by 2022. The objective of a chatbot is to emulate the conversational capabilities of humans so that when a person interacts with a chatbot they behave as if they were interacting with a peer. This is possible because the chatbot goes through a series of steps to process human data and then determine an appropriate response or action based on the user's query. There are already various examples of Artificial Intelligence (AI)-based chatbots, for example: Cleverbot, Cortana or Tay. First of all, Tay [2], Microsoft's first public experiment involving the test of a bot on Twitter, was so successful that it began to behave like its followers. Over time, however, after just 16 hours of activity it was necessary to turn it off because they she had begun to exhibit xenophobic, feminist and racist behavior. It was a similar ending for the conversation between two AI entities developed in the Facebook labs, trying to make them talk to each other, after some time they began to speak a language that was known only to them. While the epilogue was not what was expected, these experiments showed how much the technology around smart chatbots had evolved. Chatbots are a hot topic among tech behemoths like Facebook and Microsoft, as well as smaller messaging platforms like Telegram and Slack, which have made their frameworks available to developers to ensure smooth development.

A chatbot can be used for a variety of purposes, and the Internet of Things (IoT) can easily be added to this list. One of the reasons why the IoT is struggling to take off is the difficulty of less experienced users installing or configuring their devices, as well as solving small, common problems. This forces users to rely on qualified staff to fix simple problems on a regular basis, which makes the user experience frustrating. In this scenario, a chatbot can have a vital role in improving the user experience, as when properly programmed and inserted within the reference IoT platform, it would give the user the necessary support when dealing with complicated actions thus fulfilling the lack of skills. Obviously, it would not only provide some help in performing specific actions but it would also provide information that will be specifically requested by the user, such as, for example, about the status of their car, home, work and so on, significantly reducing the barriers between the user and connected objects. All in natural language (human language) rather than relying on navigation through a graphical interface in a mobile application or website. Unfortunately, progress is currently being made at a snail's pace. The truly conversational chatbot, which will be able to autonomously interpret user inputs, is still a long way off, but several research efforts are moving towards the unification of an ecosystem that is currently very fragmented. The IoT space is at an inflection point, with conversational user interfaces at the forefront. This process is becoming more achievable day-to-day with services that help companies to easily integrate natural language understanding into their products. The scope for conversational user interfaces is enormous, and it continues to expand. With a variety of technology available for the implementation of such systems, the next step is to figure out where machine learning strategies make more sense than other technologies and whether they can potentially save us time and enable people to focus on more valuable tasks.

This work will focus on the interfaces between users and IoT-enhanced environments. Indeed, the chatbot will provide users with a user-friendly interface to assist them in creating their profile and managing their services and objects. The virtualization of the user is also introduced to suggest relevant services that are expected to be the most suitable and interesting for each user based on their profile and thanks to context-awareness mechanisms. The use of chatbots in the IoT already has a history, but the main limitation comes from the fact that these are vertical and domain-specific solutions. The solution we propose, thanks also to the intermediation of user/device virtualizations, allows the reuse of the same chatbot interface on different IoT applications.
Accordingly, the contributions of the paper are as follows:

- We analyze the key requirements for the development of a chatbot for the IoT scenario and provide a description of the architectural components of the chatbot-enabled user virtualization;
- We discuss the integration of the chatbot system in a fully distributed virtualization-based IoT architecture;
- We provide the details of the implementation that have been carried out to develop a prototype;
- We present some experimental results for the evaluation of the capability of the proposed solution to identify correctly the user intention interacting in the IoT environment.

The paper is structured as follows. Section 2 discusses the major related works in this area. Section 3 presents the key requirements in designing a chatbot system in a virtualization-based IoT platform. The system architecure is presented in Section 4, while in Section 5 we illustrate a use case to better understand the reference scenario. The implementation and the experimental results are shown in Section 6 and Section 7 respectively. Finally, Section 8 concludes the paper.

2. STATE OF ART

The first entertainment chatbot was developed in 1966, it was called ELIZA [3] and was a parody of a psychotherapist who answered the patient's questions with other questions, obtained by rephrasing the patient's questions. In 1995, Richard S. Wallace built A.L.I.C.E. [4] a chatbot made entirely with open source software that uses the AIML language, child of the XML language from which it inherits extensibility, which thus allows the chatbot to hold a conversation. With the growing interest in artificial intelligence and with the idea of simplifying the interaction between man and machine, more and more companies, have developed or directed part of their research on chatbots.

In industry and in particular in the IoT field, chatbots are entering in workflows in a capillary way. This is because thanks to their characteristics they allow to stem the difficulties of configuring and troubleshooting devices encountered by operators and users.
The first problem that arises when new solutions have to be introduced into existing systems is to understand the impact in terms of complexity. To understand the complexity in [5] the authors analyzed the possibility of creating a general architecture that would allow the integration between chatbot and IoT systems in a simple way. The study found that what chatbots and IoT have in common is that they adopt their services through relatively simple, often RESTful, web APIs. In this scenario, adopting a service-oriented development approach to development, integration is feasible thanks to RESTful HTTP standards and protocols. In this case the ISO/OSI application level is the only level concerned, without having to go down to the underlying levels. It is therefore clear that with design precautions, the integration between chatbot and IoT platforms is extremely simple.
In the literature there are several examples of systems that use chatbots to interact with IoT devices. In [6] the authors implement a chatbot integrated with an agricultural plant monitoring system. In their implementation they use fuzzy logic and Natural Language Processing (NLP) to interpret user inputs. The user asks the plant a question and it answers. An orchid was used for the experiments. The success rate of the interaction between question and answer was 71%.
An interesting proposal is presented in [7], an IoT system with AI chatbots for plant monitoring capable of monitoring various parameters useful for knowing the health of houseplants. Alongside the IoT system, we implement a chatbot to inform the owner about the current conditions of the plant and its current needs. The data is also stored and through the bot the user is able to analyze the graph and determine the level of wellbeing of the plant and any problems.
In [8] an integrated Chatbot-IoT system is implemented to make the monitoring and improvement of water quality quick and efficient. For monitoring, a network of IoT sensors was created, supported by a cloud platform. Inside, a chatbot has been integrated that uses text mining techniques to interpret user inputs. The result showed excellent performance with high precision and recall for each class.
In [9] and in [10] two IoT platforms for home monitoring and remote control are presented. They have a built-in chatbot that can understand text or voice commands using NLP. Using different APIs and protocols, the authors have obtained user-friendly systems for controlling home devices. They also demonstrated how an architecture structured on multiple services is effective and easy to implement.

The authors in [11] focus on integrating chatbots and IoT to address a critical problem such as air quality awareness. In this case, the chatbot not only provides users with information on air quality, temperature and humidity, but also provides services such as subscriptions to preferred air quality monitoring points. Furthermore, advanced functions have been implemented that can be managed entirely via chat such as: alarm services, threshold settings, geoquery and advice based on pollutant levels.
In [12], a healthcare prognosis chatbot based on AI-IoT and with adaptive learning capabilities is proposed. The aim of the system is to provide medical diagnoses in real time and to support patients in the absence of healthcare professionals. The interactive system provides tools to collect data, answer general medical questions, provide assistance and provide alerts to remind patients that they need to take their medication. The system in question has shown an accuracy of 90% of the answers. Similar to the previous one, in [13] is presented a chatbot designed to increase the capacity of health services so as to reduce the management costs for medical consultancy services. Unlike the [12] proposal, this chatbot is paired with an IoT device for detecting vital signs. This combination can help people know their health status.
With COVID-19 social stress has grown exponentially, the proposed work in [14] uses a chatbot to defeat the stress of individuals during the period of isolation. This chatbot allows persons to interface with remote clinical specialists. In this case, artificial intelligence and NPL techniques combined with a clinical chatbot. This will understand if it is enough to continue the conversation with the bot or if the user needs to interact with a human professional. From what has been analyzed, it can be seen that integration with IoT systems is very useful for enabling inexperienced users to use advanced features in a simple way. Our proposal is to insert a support chatbot within a Social IoT (SIoT) platform. In this way, all the applications that will be hosted within it will be easily usable and configurable even by the less experienced.

3. BACKGROUND

Currently, there are multiple architectural solutions for IoT (vertical, horizontal, centralized or distributed solutions, etc.) with involvement at various levels of the user in interacting with devices that surround them. The design of an intuitive interface requires key requirements that best fit the chosen architectural solution. The following subsections show the technological needs in the design of a chatbot and the chosen reference IoT architecture.

3.1 Key requirements in designing a chatbot system

In human-interaction-based applications the processing time and the latency in general are key requirements. Similarly, in a chatbot application, users expect immediate responses in comparison to other web and mobile applications. The processing time should not increase directly or exponentially with the number of users, but rather should be constant and perform at its best almost regardless of the workload. To get high scalability, we can rely on serverless cloud services such as Amazon AWS, Google Cloud Platform, IBM OpenWhisk or Microsoft Azure. On these platforms, we are able to develop lightweight event-based architectures so as an event can have more than one handler and is also able to start the execution of short isolated parts of codes written in order to perform specific atomic tasks. Additionally, each event handler can create one or more event after processing the event data. Function as a Service (FaaS) is a cloud service model based on serverless architecture that allows developers to build a flexible system that fits well to pulling entire functions up and down for each request. In chat applications, the speed with which applications are instantiated is crucial to reduce latency times. In an FaaS solution, the platform manages the loads at the level of individual requests, optimizing in terms of performance and costs. However, it is not possible to implement a chatbot system entirely in FaaS, as there are other features that require other service models to ensure, for example, data persistence, back end to an IoT platform or front end for user interface rendering. And it is not recommended to use exclusively a container-based service model (Containers as a Service (CaaS)) even if currently Kubernetes, at the level of scaling, is approaching FaaS solutions thanks to intelligent traffic management based on analysis models that imply FaaS features. Based on the application context, however, we can think of a hybrid use of containers and FaaS, which is the solution we adopted in our system.

The use of a pay-per-use model reduces operating costs compared to a traditional system that requires the allocation of the resources of one or more processing instances. In fact, the FaaS model allows you to activate the necessary functions on request and to release them immediately after the execution of the tasks. So we can see that, given the speed with which requests must be processed and given the conditions of traffic non-uniformity that make it impossible to estimate the users who will actually request the service, the most convenient solution for the implementation of the chatbot is the serverless one. In addition to scalability, it is also necessary to pay attention to the latency at start-up, that is the time that a FaaS function takes to respond to requests. Typically in all the platforms mentioned they take from a few milliseconds to a few minutes, this time is variable and depends on various factors, such as programming languages, for example.

Another key requirement is the management of the state, a serverless system by its nature is stateless, to overcome this problem we will rely on an instance of a database that will take care of saving both the state and further inputs that will be used to manage the requests.

3.2 Reference IoT architecture

Recently, several studies have looked at the problems of managing and effectively using large numbers of heterogeneous devices, and have found a solution in the use of social networking principles and technologies. In [15], the definition of the Social IoT (SIoT) has been formalized, and it is intended to be a social network in which each node is an entity capable of forming social relationships with other things on its own, according to the rules set by the owner. The proposed model is based on the Lysis cloud SIoT architecture [16], which incorporates virtual objects as digital counterparts to physical objects to enhance their capabilities in a transparent manner to users. Lysis architecture foresees a four level structure of independent modules. Its lowest layer is populated by Real World Objects (RWO), i.e. physical IoT devices able to perform basic tasks. On top of this, the virtualization layer, directly interfaces with the real world and is populated by Social Virtual Objects (SVO), which are VOs with socialization capabilities. The aggregation layer is responsible for composing several SVOs into entities with extended capabilities, called Micro-Engines (MEs). Finally, at the application layer, user-oriented macro-services are provided (APP).

Socialization algorithms implemented in the first two levels allow for the creation of social relations as foreseen in the SIoT paradigm. The resulting social graph is exploited to find the required resources.

3.3 User virtualization in IoT

The widespread presence of connected objects throughout daily life has allowed the Internet of Things (IoT) to spread. The IoT vision forms a collaborative ecosystem for a multitude of heterogeneous objects with different connectivity and computing capabilities to achieve the common purpose of providing user services.

At the current time, the IoT platforms seem to present several pending issues that prevent a full spread of IoT applications. Indeed, services are mostly configured manually by users, according to preferences that could be shared among similar or cross-domain services (e.g., preferences about ambient temperature at home and at work to manage HVAC systems). Secondly, the users that access an IoT platform need to autonomously look for the required services among a plethora of them. The risk is in a decrease in the quality and reliability perceived by users, who therefore risk being discouraged from using IoT applications. Our system has the objective of exploiting the concept of Virtual User (VU)[17] to improve the user experience and, at the same time, enhance the efficiency and usability of the IoT platforms and services.

The VU is the virtualization of a user, and it is represented by an agent that enables the following major benefits: providing users with a user-friendly interface that enables automatic or assisted setup of their profile, objects and services; proposing the services that are expected to be the most suitable and interesting for each user, based on their profile and context: and enabling objects to be as much plug & play as possible.

The specific focus of this work will be on the interfaces between the users and IoT enriched environments. Indeed, a user-friendly interface will be provided by means of the chatbot to users to assist them to create their profile and manage their services and objects. Based on their profile and thanks to context-awareness mechanisms, the VU will be able to suggest relevant services that are expected to be the most suitable and interesting for each user, and settings will be automatically configured.

4. PROPOSED ARCHITECTURE

The proposed solution is aimed at designing and experimenting a chatbot system that simplies the interaction of the users with the VU in an IoT platform by means of text messaging. As previously explained in Section 3, the VU is the virtualization of the user and takes decisions on their behalf for known activities; as such, it interacts with all the modules of the Lysis IoT architecture [17]. The VU was not introduced specifically for the Lysis platform. In fact, it follows the more general concept of virtualization in the IoT and of virtual objects. The concept of VU arises from the need to provide a virtualization element that constantly deals with the context of the user it represents, their interaction interfaces and their IoT services. In this scenario, the VU is a standalone element in a distributed virtualization system. The Lysis platform, which we use as a development environment, is precisely a distributed system of elements that allows for the creation of a social network among virtual objects in order to facilitate their interaction. The VU could be used in a centralized system, possibly vertical; however, in this case it would not bring all the advantages that characterize the implementation in a distributed system. Furthermore, any IoT platform that is a candidate for the integration of the VU and its interfaces, such as the chatbot, should provide open APIs that allow for full integration.

Fig.1 shows the components of the overall architecture according to the Lysis model. The VU communicates with all levels to provide user preference information to build tailored IoT services. The chatbot system is a back-end service for proxies the communication between the users the the VU.

Fig. 2 shows the architecture of the proposed solution that has been designed to address the requirements that have been previously discussed.

The upper layer implements the functionalities to receive and transmit requests and data. The requests are generated by either web apps or (proprietary / non-proprietary) messaging services which are used by the user for sending and receiving messages. Each request contains the *intent*, i.e., the action that the user would like to take, which is written in natural language. The intents are then received by the chatbot API gateway to be sent to an AI-based service that interprets the intents to

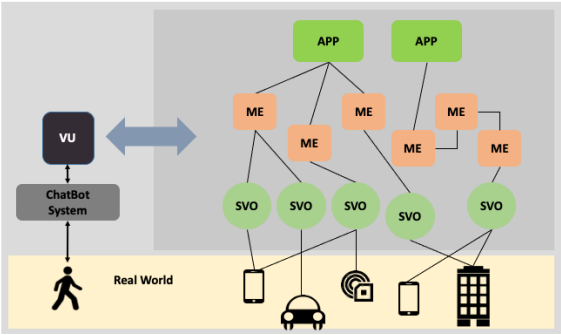

Fig. 1 – ChatBot-enabled VU in the Lysis architecture

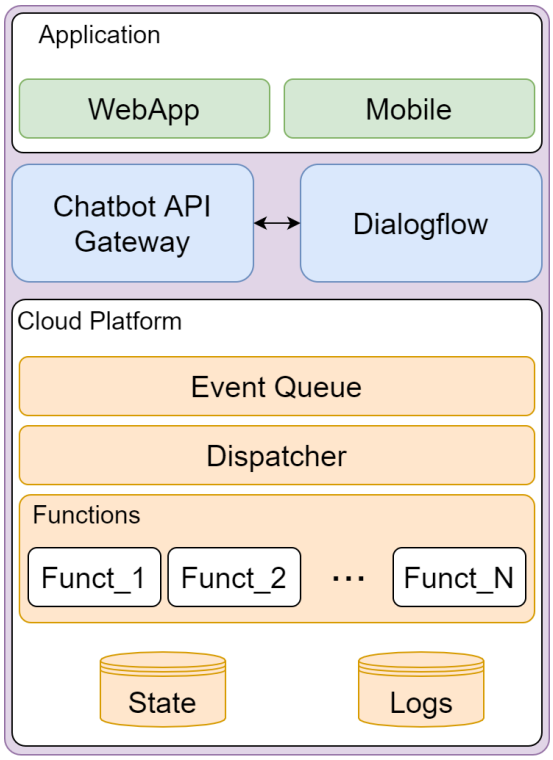

Fig. 2 – The propose chatbot architecture

understand what events are associated with these. In particular, in the developed solution, we have adopted the Dialogflow service. The resulting event is then communicated to the gateway (through the edge layer), which takes care of sending it to the interface that then stores it in an event queue waiting for the dispatcher to direct it to the needed function. Indeed, there is a different function for each event or group of events that is activated when a given event is received. At this moment, a search is then carried out to understand if there is already an active instance for this function; if not, a new instance of this function is activated, the event status is temporarily saved and then finally the function instance is destroyed once it is no longer needed. The functions to be implemented do not have to be all accessible via a gateway route; in fact, it will be sufficient to make them accessible via a specific topic. In this way it is easier to manage the planned events also considering the interaction between the same functions. The gateway is an HTTP server in which routes and endpoints have been defined, where each route is associated with a FaaS function. When the gateway receives a request, it identifies the corresponding routing configuration by calling the relevant FaaS function. Fig. 3 shows the gateway workflow. When the user at a given time needs to request information, they will forward a message to the gateway. Herein, let us assume that the user is already authenticated; at the time of sending the request, a POST call is made to the URL / API / createHook. This URL takes care of creating the WebHook to use for communicating (conversing) with the cloud functions. The WebHook consists of two basic parts: a token and an event. The token is generated during the creation of the WebHook and is used to authenticate the communication, lasting for a predefined amount of time; if this cannot be authenticated the call is stopped. The event, on the other hand, becomes the topic. Whenever the endpoint receives signed data from the chat service correctly, the gateway has to respond immediately with an HTTP status code; if everything went smoothly, it generates a code 200 (OK), 201 (created) or 202 (accepted). However, if the data is not signed correctly or even the signature is missing, it responds with a 403 code (forbidden) and does not provide the broker with the needed data. To protect transactions between servers, it is convenient to use SSL / TLS.

This described architecture is integrated in the Lysis IoT platform so that the outcome of the functions is taken by the VU to perform the resulting tasks. The following are the different needed functions: Message Handler; Action Service; Save Conversation; Send Message; Failure Handler; and Save Logs. These are briefly presented in the following.

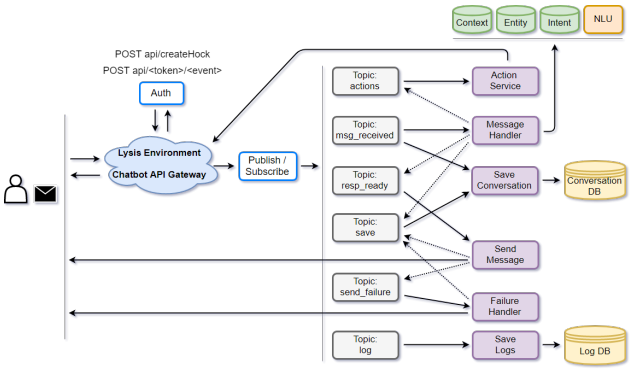

Fig. 3 – Process flow for the whole system

4.1 Message Handler

If the gateway gives the green light, it stores the input in an event queue and then processes it as soon as possible. The various topics provided allow for calling the related functions. In the case of a request generated by the user, then the topic will be *msg_ received* which calls the Message Handler function; this is the only function that can be called directly from the gateway interfaces. This function processes the request and creates the output to be provided to the user. The construction of the output takes place in various stages and on the basis of the user's request.

As a representative case, let us consider the user that wants to enable remote surveillance of her home (at the moment this functionality is off). The user sends the following text "Set video On"; in order to understand what to do with this command, the message must be broken down and analyzed. From here, therefore, the function has to understand the entity, the user intent and the context using an NLU (Natural Language Understanding) algorithm. Obviously, in order for the analysis to be relevant to what is requested in the messages, the system must be able to store and analyze the status otherwise, if this was not the case, the responses and actions taken would not be relevant to the general context of the conversation. On the basis of the results obtained then, if an intervention or a reading is required in a given device, you will have to be able to invoke, through an appropriate topic, a function that will implement or request what the user needs and then return it to the Message Handler. Once all the data has been obtained, it is possible to create the reply and make it available for sending.

4.2 Action Service

When the Massage Handler function needs data that is on the platform, it must be able to retrieve it. The simplest way is to rely on a second function with this objective. The Action Service function takes care of retrieving the requested data. Once the Message Handler has processed the request and on the basis of the NLU algorithm has understood the actions to be taken, it activates the Action Service which fetches to the platform requesting data or making settings. In the event of an error, the identification code will be returned.

4.3 Save Conversation

This function is invoked when the save event occurs. This can be invoked by the Message Handler, Send Handler, and Failure Handler functions. In the first case, as soon as the message is received, this (in addition to being taken over by the Message Handler function) also passes through the function in question which will create a record containing the request and the status of the conversation. The second case is similar to the first but now it takes care of saving the response produced by the Message Handler function; however, if the sending fails, the save event cannot be invoked and *send_failure* will be invoked in its place. The last case is equal to the second except for the fact that now failing or not, a record is still created which will be the message produced in the case of success or an error message in the case of failure. Each time you save the conversation, the status is also saved.

4.4 Send Message

The sending function is the one that takes care of forwarding the response to the user through the chosen messaging service. It could be for example a telegram rather than a proprietary application created ad hoc.

Whatever the application chosen, this function first of all sets up the WebHook and then, if the setting is successful, sends the response produced to the user; if the setting fails, it contacts the Failure Handler function to manage the mistakes. A best practice is to use separate functions and topics for receiving, error handling and sending. This way there won't be problems in contacting the correct endpoints; accordingly, operations such as save and retry won't be taken over by this function.

4.5 Failure Handler

If the topic becomes *send_failure*, it means that there was a problem sending data to the user. To manage these types of problems there is the need to rely on a special function. When the Send Message function fails the first attempt, it contacts the Failure Handler function passing the message and the error code returned by the attempted send. The function is encoded in order to retry the sending for a certain number of times after which the user will be notified that there is a problem in satisfying their request. If, on the other hand, the sending is completed within the established number of attempts, the message is delivered in a totally transparent way to the user.

4.6 Save Logs

This is used to archive all messages related to the system in general; in this way it is possible to monitor the flow and see if there are any problems or if some parts need some actions to be performed. This is a feature that can be implemented by relying on the logging of activities of the cloud platform. In addition, some platforms such as AWS or Google Cloud Platform allow for saving in the log file, in addition to the default entries, new fields at the user's discretion. In so doing, by integrating the system logs with those of the requests to the bot, it is possible to have complete and detailed logs.

5. SCENARIO

The IoT Lysis platform currently does not provide any help for the user either with regard to the deployment of SVO or with regard to the resolution of any problems such as failures, unresponsive devices and so on. To simplify the user-platform-SVO interaction, the intention is to insert a bot within the platform that guides the user in carrying out those activities that are currently cumbersome or even impossible to perform remotely:

- SVO deployment
- Problem resolution
- Inquire of devices
- Setting
- Task automation

To create the bot, the Google Cloud platform was chosen, in which all the back end and the various functions are hosted, alongside the DialogFlow service provided by Google that offers a retrieval-model-based technique for matching responses with the aid of machine algorithms learning, where the latter can be enabled at the user's discretion. This function, if enabled, allows us to have some flexibility in the interpretation of the user's requests as the answers are given based on the best score obtained from a classification prior to the choice of the answer. In this way, therefore, it is possible to manage any spelling or form problems that might cause errors in recognizing the correct intent for the request made.

The proposed solution allows for a dynamic composition of the services that can be provided, given the ability of the bot to query any SVO owned by the user present on the platform. When the user queries the chatbot, they will be offered various choices and based on the SVOs that are selected, a service is composed with only the choices made by the user. For example, in the car, in addition to the SVOs relating to the car, you may also need SVOs relating to other environments, for the purpose of continuous monitoring, the service offered by the bot therefore includes the data from these SVOs. In addition, you would also have the possibility to save them and re-propose them at a later time as favorite services. Fig. 4 shows the sequence diagram which illustrates the simple steps that take place when a request is sent to the bot.

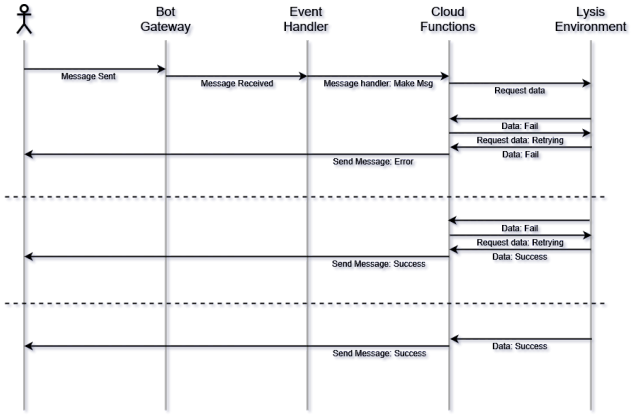

Fig. 4 – Use case diagram

When the user accesses the chatbot interface, they are presented with the various options. Once the desired option is selected, the bot will send a message to the bot gateway, who will take care of handling the request, labeling it and sorting it to an Event Handler (EH).

The Message Handler will recognize that a message has arrived and delivers it to a cloud function that takes care of the part of creating the response message. Then, it has to collect the data in addition to the textual answers by querying the platform that contains the SVOs necessary for the composition of the answer. At this moment, we may have two different scenarios that we analyze below:

- The data request fails
- The data request is achieved

We can see in Fig. 5 the workflow of requests in a possible user interface. All these interventions are immediate, the most expensive response in terms of timing is the one in which the data is requested, in this case the video stream. But in principle, the time between a send-reply is given by the user interaction time with the device plus the delay introduced by the bot to reply, which is a maximum of a few seconds.

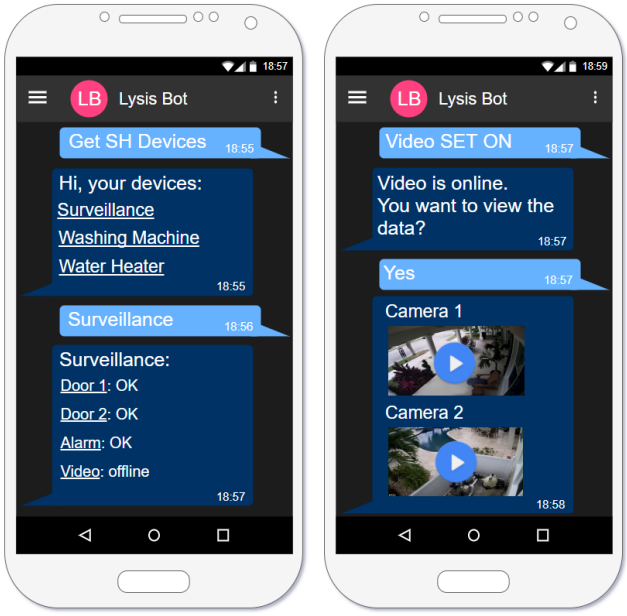

Fig. 5 – Representative flow of messages exchanged between the user and the bot

We have implemented the chatbot system that allows for interacting with the platform and for setting and sending requests to the devices. The queries to the bot are made in natural language and are taken over and processed by the DialogFlow platform, with has been integrated in our system. The messaging system selected has been the Telegram messaging client which is used by the user.

The development of the bot was divided into two parts: design and development of all the components necessary for the NLU functionalities; development of the gateway and the functions necessary to handle the events and associated data in the chatbot platform. An agent has been created within the DialogFlow platform. Agents are NLU modules that deal with transforming user requests, expressed in natural language, into *usable* data, i.e., data that can be associated to actions to be activated. To ensure that the requests are interpreted correctly, all the possible *intents* and *entities* have been loaded into the agent. Intents are JSON files and have been designed and built in order to map the user's requests with the actions to be performed in the best possible way. In order to have the best match between request and intent, new entities (all synonyms for a given word are associated to an intent) have been developed, in addition to those made available by the platform, which were able to best characterize the IoT context

of use we needed. Therefore, various attributes have been created within each entity, with their synonyms, in order to be able to extract the values of the input parameters without these being necessarily identical to those we had foreseen in the phase of creating the intents. In this way, every time an intent is activated, the platform will return a JSON file with the information on: Intents; Action; Event; Response; Contexts; Parameters; Score.

At each request, these parameters are updated based on the intent that is activated at that time so that, at the next call, the new intent to be activated is also chosen based on the previous parameters. In this way, it is possible to completely contextualize the conversation. In fact, the contexts section contains all the active contexts in that call ordered according to their lifetime. With this mechanism it was therefore possible to implement a management of the state of the bot allowing for the exchange of variables between subsequent requests.

6. IMPLEMENTATION

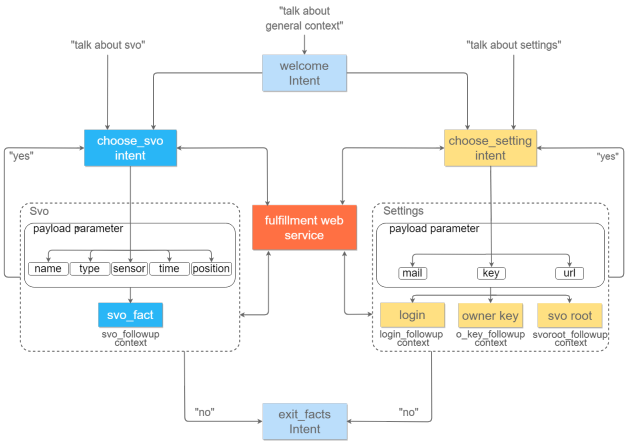

Fig. 6 – Flow diagram of the bot

As we can see in the diagram in Fig. 6, it is possible to enter one of the intents based on what is asked. Let's refer to our use case previously described related to the surveillance. As we said, the user has to ask the bot for sending the video stream from the video camera device; if it is turned off, it will first be asked to turn on and then send the data. When the user needs to use the bot, they have to log in and then send the message "show me what happens at home". This activates the "svo_facts" intent through the "svo_facts" event and setting "svo _followup context" with a certain lifetime set as the current context. The bot then responds by giving the list of objects that are indexed in their home. By selecting the video surveillance, the user remains within the "svo_followup context" and then activates the loop indicated with "yes" in the diagram; by reactivating the svo_facts intent, the context is updated again and the bot responds by displaying the state of the object. On the basis of this, then it allows for the choice whether to activate it or not. Once concluded, if the user decides to perform different actions to the question "do you want to do other operations?" answering "no", they activate the reset of the contexts and the "exit_facts" intent, initializing the bot for new requests.

7. EXPERIMENTAL RESULTS

The experiments have been conducted to assess how effective the chatbot was in understanding the user requests and perform action accordingly. In the following section we describe the performed tests with reference to the access to the platform and perform device setting and request data of interest. The performance of query matching results has been also analysed.

7.1 Access to the platform

Fig. 7 shows the interaction with the chatbot with the intent of accessing the Lysis platform. The figure shows the flow of questions and answers between the bot and the user. Remembering that it is necessary to authenticate, to be able to use both the chatbot services and to have access to the resources made available by the platform, the first question asked was on how it was possible to authenticate. The bot's response was a message with the instructions on how to log in and, once logged in, it suggested to the user that it was necessary to enter some additional information to complete the configuration. The user then asked how to enter the owner key and the SVO root, to which the bot answered by providing a description of where to find them and information on how to enter this information. This was possible thanks to the fact that when you ask for information either on the key or on the root SVO, the respective context is activated allowing you to keep track of what was previously requested. After completing the configuration and logging in again, you can see that the welcome message is simply given, a sign that the configuration was successful.

7.2 Setting the devices and data retrieval

We also tested the ability to request data from the platform and apply the desired settings to the available devices, all with the most natural language possible. Fig. 8 shows how it was simple to request the list of devices and their current status. If you want to switch a device on or off, simply specify which of these actions should be applied and the setting will be performed. The request for data was also handled in a similar way; in the sentence it is just needed to specify which data is needed and from which environment to obtain the requested data.

7.3 Elaboration time and latency evaluation

The time spent processing the information sent to the chatbot was very low. This happens thanks to the use of an ML engine that is fully running in external services and for the efficient implementation of the sample questions on the platform. This allows for a low latency between a request and the response and this makes the user

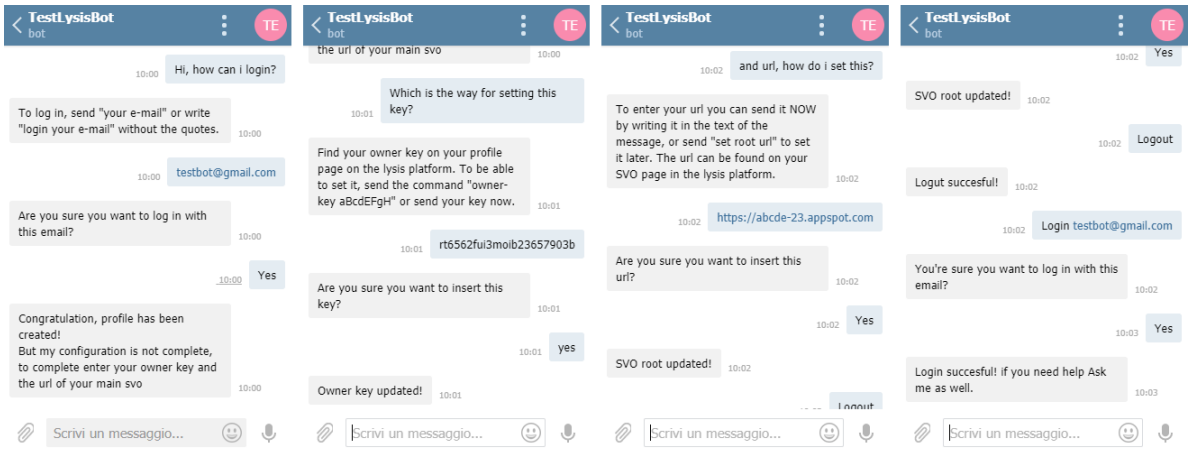

Fig. 7 – Bot tests: access to the platform

experience a smooth interaction. For instance, using a virtual machine instance with 1vCPU and 512 Mb of RAM, the latency values of the system to process and provide a response for a single request are between 100 ms and 200 ms, to which the delay introduced by the network should be added. This brings to an overall round trip time of less than a second. Furthermore, thanks to the ductility of the serverless system, by appropriately configuring the load balancing rules, when all the service instances are occupied a new instance can be started to automatically lighten the load of others.

7.4 Analysis of the questions matching scores

We have also analyzed the relevance of the questions submitted to the bot with the patterns inserted in the intents, created on the Dialogflow platform. The score calculated by DialogFlow was used for this purpose. This evaluates the level of confidence of the question submitted to the bot with the example ones present in the platform. This confidence level is calculated based on the state of the conversation and exploiting the Term Reinforcement techniques. These techniques allow for a greater weight to certain words through their repetition or the use of synonyms. Score values range from 0.0 (completely uncertain) to 1.0 (completely certain). In the proposed implementation, once a question is evaluated, there are two possible outcomes: a) if the question achieves a confidence match score greater than or equal to the classification threshold setting, the higher confidence intent is triggered; b) if no intent meets the threshold, no match is returned. In this case the threshold was set to 0.7. The score plotted in Fig. 9 and Fig. 10 indicates the quality of the match between the ideal question (the one contained in the intent) with the real question (the one generated by the user). Obviously, the sentences inserted within the intent are constructed, with the help of the entities, in such a way so as to be as general as possible, so they are not strictly meaningful sentences but rather they are composed only of the words actually necessary to give a meaning to the sentence so as to be able to guarantee the best match even with requests that are not well formulated, albeit with a lower score than the optimal one.

In Fig. 9 we can see the scores of the flow of requests that have been submitted to the bot during the configuration phase in two distinct cases. The first case, called “Best”, was produced by submitting to the bot the sentences formulated as similar as possible to how they were inserted into the intents, trying to make them as close as possible to natural language. The second case, called “Worst”, on the other hand was formulated using the synonym of the keywords and looking for a grammatical form quite different from the one used in the previous case. Similarly, in Fig. 10, the same analysis was performed for the second test, where device setting and data request were performed. Sentences 4 and 7 in 9, sentences 4 and 10 in 10 are cases in which the match between sentences is not accurate. This phenomenon is governed both by the number of synonyms that have been associated with the entities, and by the level of similarity between the various intents implemented and their length. For example, if you have two intents that trigger two different events but are very similar in natural language, the classifier will be less accurate about which one to choose. In Table 1 we also show the average values which demonstrate that there is not a big difference between the “Best” and “Worst” cases; indeed, in both cases it was possible to configure the bot, request data and set the devices smoothly without any issue about possible request misunderstanding. Obviously, the better the intents are constructed, the easier it will be to get accurate matches by submitting questions that are apparently different but express the same concept.

Table 1 – Comparison between the average values of the scores obtained for the two considered scenarios

	$\overline{S}_{Best}$	$\overline{S}_{Worst}$
Platform access	0.956	0.844
Device configuration	0.925	0.819

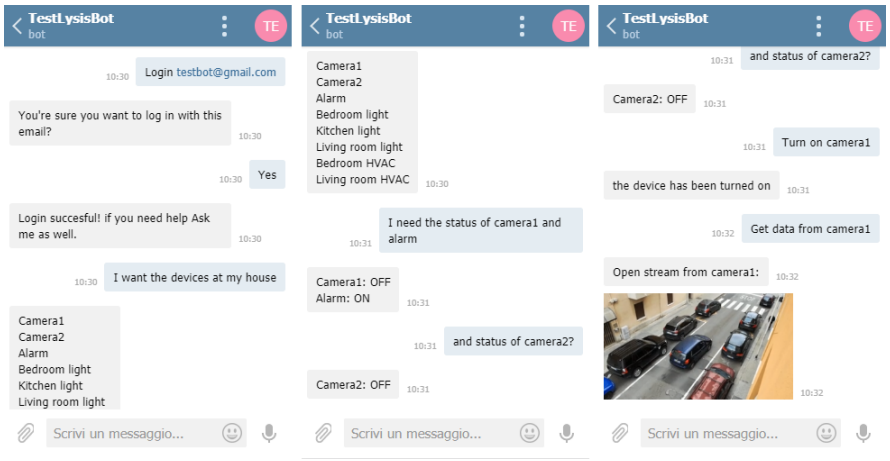

Fig. 8 – Bot tests: request of data and setting of the device

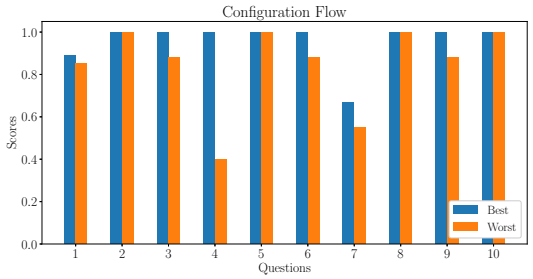

Fig. 9 – Scores that have been obtained by matching the queries with the intent during the platform accessing activities

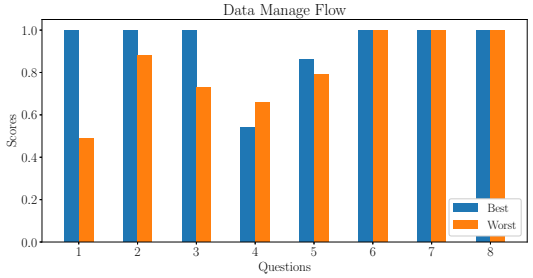

Fig. 10 – Scores that have been obtained by matching the queries with the intent during the setting of the devices and data retrieval

8. CONCLUSIONS

This study has investigated the possibility of integrating a virtual assistant, developed in the form of a chatbot, within an IoT platform to help and guide the user to easily carry out the various operations that would otherwise be cumbersome and sometimes complicated. This need, as we know, derives from the fact that the configurations and requests for data, for an inexperienced user, are not immediate but may require various steps to be completed and may be frustrating.

A bot has been then developed which, thanks to a natural language understanding engine, is able to process the user's requests formulated in a natural language. The bot essentially works as a mediator between the real world and the virtual world. In the experiments that have been carried out it has been possible to see how simple it is to configure the bot and use it to interact naturally with the IoT platform. We specifically focused on platform access, device setting and data request. It has to be said that for these experiments the operations carried out were simple but still encouraging for future developments. One of the most interesting actions is certainly the ability to deploy applications quickly and easily as well as being able to use the bot as a guide for troubleshooting, knowing in real time if the various devices are faulty or malfunctioning, so as to restart them automatically.

ACKNOWLEDGEMENT

This work has been partially found by the POR FESR Sardegna 2014 with the project Farmainforma (RICERCA_1C-38).

REFERENCES

[1] i. Gartner. 2020. URL: https://www.gartner.com/smarterwithgartner/top-cx-trends-for-cios-to-watch/#:~:text=Chatbots%2C%20virtual%20assistants%20and%20robots,up%20from%2015%25%20in%202018..

[2] G. Neff and P. Nagy. "Automation, Algorithms, and Politics Talking to Bots: Symbiotic Agency and the Case of Tay". In: *International Journal of Communication* 10.0 (2016).

[3] J. Weizenbaum. "ELIZA—a Computer Program for the Study of Natural Language Communication between Man and Machine". In: *Commun. ACM* 9.1 (1966), pp. 36–45.

[4] R. Wallace. "The anatomy of A.L.I.C.E". In: 2009, pp. 181–210.

[5] R. Kar and R. Haldar. "Applying chatbots to the internet of things: Opportunities and architectural elements". In: *arXiv preprint arXiv:1611.03799* (2016).

[6] S. Wiangsamut, P. Chomphuwiset, and S. Khummanee. "Chatting with Plants (Orchids) in Automated Smart Farming using IoT, Fuzzy Logic and Chatbot". In: *Advances in Science, Technology and Engineering Systems Journal* 4 (2019), pp. 163–173.

[7] M. H. A. Fadzil and D. Ab Kadir. "Plant Monitoring with Artificial Intelligence Chatbot". In: *Journal of Computing Technologies and Creative Content (JTec)* 5.2 (2020), pp. 34–38.

[8] M. U. H. Al Rasyid et al. "Integration of IoT and chatbot for aquaculture with natural language processing". In: *Telkomnika (Telecommunication Comput. Electron. Control* 18.2 (2020), pp. 640–648.

[9] G. Alexakis et al. "Control of smart home operations using natural language processing, voice recognition and IoT technologies in a multi-tier architecture". In: *Designs* 3.3 (2019), p. 32.

[10] S. Ahmed et al. "Smart Home Shield and Automation System Using Facebook Messenger Chatbot". In: *2020 IEEE Region 10 Symposium (TENSYMP)*. IEEE. 2020, pp. 1791–1794.

[11] S. Mahajan et al. "Design and implementation of IoT-enabled personal air quality assistant on instant messenger". In: *Proceedings of the 10th International Conference on Management of Digital EcoSystems*. 2018, pp. 165–170.

[12] J. E. P. Reddy et al. "AI-IoT based Healthcare Prognosis Interactive System". In: *2020 IEEE International Conference for Innovation in Technology (INOCON)*. IEEE. 2020, pp. 1–5.

[13] K. Sivaraj et al. "Medibot: End to end voice based AI medical chatbot with a smart watch". In: 9 (2021), pp. 201–206.

[14] C Balasubramaniam, S Velmurugan, and M Saravanan. "DESIGN AND DEVELOPMENT OFSMART HEALTHCARE CHATBOT APPLICATION USING AI-ML". In: *Journal of Natural Remedies* 21.7 (S1) (2020), pp. 13–20.

[15] L. Atzori, A. Iera, and G. Morabito. "SIoT: Giving a Social Structure to the Internet of Things". In: *Communications Letters, IEEE* 15 (2011).

[16] R. Girau, S. Martis, and L. Atzori. "Lysis: a platform for IoT distributed applications over socially connected objects". In: *IEEE Internet of Things Journal* PP.99 (2016), pp. 1–1.

[17] R. Girau et al. "Virtual User in the IoT: Definition, Technologies and Experiments". In: *Sensors* 19.20 (2019), p. 4489.

AUTHORS

Raimondo Cossu is a telecommunications engineer. As soon as he graduated he did an internship at Avanade SRL where he acquired IT skills. After the internship he was hired as a collaborator in the MCLab DIEE laboratory at the University of Cagliari, where he currently works. His field of research and development is focused on the Internet of Things, cloud computing and distributed systems. Currently he is also CTO in WiData SRL, a company that deals with data analysis.

Roberto Girau is a research fellow at University of Bologna, Department of Computer Science and Engineering since 2021. He received an M.S. degree in telecommunication engineering and his Ph.D. degree in electronic engineering and computer science from the University of Cagliari, Italy in 2012 and in 2017, respectively.
From 2012 to 2020, he worked as researcher at the Department of Electrical and Electronic Engineering of the University of Cagliari, developing an experimental platform for the social Internet of Things.
His main research areas of interest are IoT with particular emphasis on its integration with social networks, software engineering, smart cities and cloud computing.

Luigi Atzori is Full Professor at the Department of Electrical and Electronic Engineering, University of Cagliari (Italy) and Research Associate at the Multimedia Communications Laboratory of CNIT (Consorzio Nazionale Inter-universitario per le Telecomunicazioni). His research interests are in multimedia communications and computer networking, with emphasis on multimedia QoE, multimedia streaming, NGN service management, service management in wireless sensor networks, architecture and services in the Internet of Things. He has been the associate and guest editor for several journals, included: ACM/Springer Wireless Networks Journal, IEEE IoT journal, IEEE Comm. Magazine, the Springer Monet Journal, Elsevier Ad Hoc Networks, and the Elsevier Signal Processing: Image Communications Journal.