Integration of the Alexa assistant as a Voice Interface for Robotics Platforms

A. Hidalgo-Paniagua[2], A. Millan-Alcaide[1], J.P. Bandera[1], and A. Bandera[1]

[1] Universidad de Málaga, home page: http://www.uma.es
{amillan, jpbandera, ajbandera}@uma.es
[2] Altran Innovación S.L, home page: https://www.altran.com
ahidalgopaniagua@altran.com

Abstract. Virtual assistants such as Cortana or Google Assistant are becoming familiar devices in everyday environments, where they are used to control real devices through natural language. This paper extends this application scenario, and it describes the use of the Alexa assistant from Amazon through an Echo dot device to drive the behaviour of a robotic platform. The paper focuses on the description of the technologies employed to set such ecosystem. Significantly, the proposed architecture is based, from the remote server to the on-board controllers, in Low-Energy (LE) hardware and a scalable software platform. This approach will ease programmers integrating different platforms, e.g. mobile-based applications to control robots or home-made devices.

Keywords: virtual assistant, human-robot interface, voice commanding

1 Introduction

Robotics has made significant progress towards full and shared autonomy in complex tasks, such as navigation and manipulation. Moreover, they are now capable of social interaction with human users, presenting a new opportunity to provide individualized care/assistance/help. A great deal of attention and research is dedicated to assistive systems aimed at promoting ageing-in-place, facilitating independent living in ones own home as long as possible [2]. The success of these systems depends on their acceptability, that can be described in terms of attitude, usability and confidence. Under these premises, it is clear that these robots require to provide natural communication channels to be commanded. However natural, intuitive interaction is usually not easy to establish, specially when engaging a population group (elderly people) in which physical and cognitive limitations are more common, and the use of new technologies, less frequent [3]. One solution to smooth this engagement could be to include a virtual assistant on the *loop*.

A virtual assistant is a *conversational, computer-generated character that simulates a conversation to deliver voice- or text-based information to an user via a web, kiosk or mobile interface* [1]. Voice-user interface (VUI) employs different technologies to recognize and process natural language, enabling users to

interact with an artificial agent by just speaking. Similar to the scenario desired by robotics assistants, the expected result is a more human-like, natural form of user interface. On the other hand, it is also important to note that voice assistants in particular, but also robotic controllers in general (e.g. the Jibo robot[3]), are becoming smaller, with lower energy consumption, and require less computing power to operate. This fact is causing voice assistants to be bursting strongly into the world of IoT (Internet of Things). This paper proposes the use of a virtual assistant as the interface between the human user and the robotic platform. The advantage of using this scheme is twofold. On one hand, the robust voice-based framework of the virtual assistants provides a natural mechanism for interaction, improving the relative low speech recognition rates (specially in real environments) that can provide previous recognition alternatives. On the other hand, the effort to introduce virtual assistants at home may take hand in hand the simultaneous introduction of robotic assistants, able of performing tasks that require manipulation or/and navigation (i.e. agents that go beyond the mentioned Jibo, which is quite similar to a virtual assistant). The integration with the *non-written* guidelines of IoT is considered in our proposal. Thus, it is built over an architecture that, from the remote server to the on board controller, is based in LE (Low Energy) embedded systems (e.g raspberry pi or odroid platforms) and is fully scalable and compatible with other technologies, as mobile–based applications for controlling robots or any home–made device.

The paper is organized as follows: Section 2 introduces the software architecture as a block diagram that describes the information flow through all components. Section 3 summarizes the process of natural interaction using voice assistants and how they allow us easily programming new applications in an abstract way. Section 4 shows the set of tools and frameworks for programming voice-based applications. Section 5 comments the hardware used in experiments, from the remote server to the on board controllers. Section 6 shows a real VUI implemented to control a robot using the Amazon's Alexa voice assistant. Alexa has been used to interface robots such as the Lynx[4] one, and is fully integrated in the ES-2A[5] robot. In our scheme, the voice assistant and the robotic platform are included within a heterogeneous and distributed system, which is open to consider other items. Finally, Section 7 draws the obtained conclusion of our proposal and the possible future works.

2 Overview of the proposal

Figure 1 shows the overview of the software architecture. The main goal is to set a communication channel between the user and the actuation devices. This channel should include the ability to have a conversation with the user, providing automatic responses to their entries if required (i.e. a chat bot). In our case, the channel has two clear end parts:

[3] https://www.jibo.com/
[4] https://ubtrobot.com/products/lynx
[5] https://espabot.es/robot-social-es-2a/

– The interaction channel with the user: composed by the Alexa assistant, embedded in the Amazon Echo Dot device, and connected through Internet to the Alexa Voice Service (AVS), that runs on the Amazon Web Services (AWS) ecosystem[6].
– The devices in charge of running the required commands, that may be robots or other devices.

Within both end parts, the third part is a remote server, in charge of managing the interaction. Messages will be managed using a publish/subscribe model. The open-source Eclipse Mosquitto[7] broker (MQTT protocol[8]) was chosen for carrying out this channel. Mosquitto is lightweight and very suitable for use with low–power boards. Finally, the required chat bot is implemented using the RedBot platform, based on Node-RED. All technologies will be reviewed in Section 4.

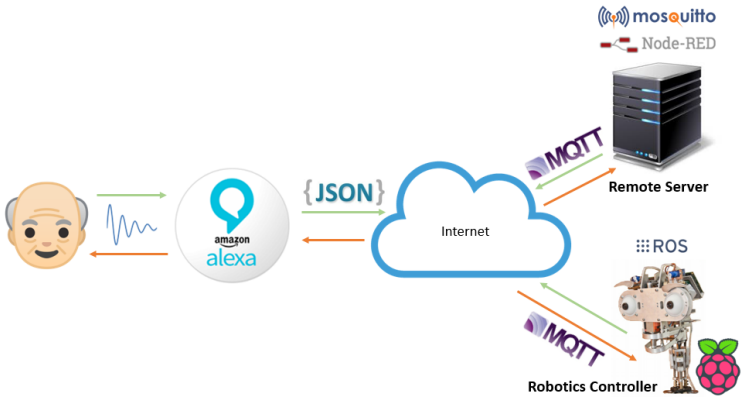

Fig. 1: Overview of the proposed system architecture

3 Natural Language Processing (NLP) using the Alexa voice assistant

Natural Language Processing (NLP) is a field of computer engineering and artificial intelligence (AI) that focuses on voice-based natural interactions between humans and computers. While research in NLP started in the early '50s, results have been traditionally constrained due to the complexity of getting useful data from voice. The problem has not a trivial solution and, in fact, in order to use voice for human-robot interaction, several stages related to different problems

[6] https://developer.amazon.com/de/alexa-voice-service
[7] https://mosquitto.org/
[8] http://mqtt.org/

must be solved. The first stage, ***speech recognition***, deals with the translation of spoken language into text. This process is known as Speech To Text (STT), and it provides a text output that allows a computer processing the potential data contained in voice in a machine-readable format [4]. The second stage is in charge of solving the ***natural language understanding*** (NLU). The aim of the NLU is to post-process the text coming from the STT, to interpret and understand the meaning of a user query or sentence in a non-structured format, and to convert it into an structured representation [5]. In short, NLU tries to understand a whole situation context instead of understanding individual words or phrases. The last stage is known as ***natural language generation*** (NLG). The NLG aims to generate natural language automatically mainly using structured text (although it is also possible to generate natural language from voice). Thus, NLG is the reverse stage of the speech recognition phase, so it is also known as Text To Speech (TTS).

3.1 NLP in the Alexa assistant

Alexa assistant tackles the problem of NLP in an easy way for programmers. Regarding STT, the Amazon's Alexa assistant implements a cloud-based STT engine that delivers the translated voice through its application programming interface (API). But the process is transparent to the programmers: all they need to do is to associate syntactical structures with actions. The syntactical structures correspond with voice translations, the so-called ***utterances***, and the associated actions are called ***intents***. The Amazon's AI engine is designed to learn as the users use the assistant, so the STT becomes better with time. This is the reason why Amazon stores the user queries: to continuously train its AI engine with new data.

After the voice is translated, Alexa assistant allows programmers to extract useful information from utterances. This is how Amazon brings programmers the NLU capabilities. To extract specific data from utterances, Alexa uses the concept of ***slots***. Slots are variables of a defined type that programmers can embed into utterances, so these ones look like regular expressions or patterns. Alexa assistant only fires an intent when a translation perfectly matches with an associated utterance. In that moment, the assistant delivers to the rest of the system the specific data extracted from utterances through slots. Once an intent is executed, Alexa can deliver a customized spoken feedback to the user. This feedback is generated from text data in a NLG phase. This is a recommended, but optional, feature of this kind of smart devices. The major difference with other competitors is that Alexa assistant eases this work. The Amazon's assistant communicates outside using JSON-formatted messages, and to include a spoken feedback programmers, it only needs to embed their customized messages into the original message received from Alexa and returns it back.

Thus, the whole NLP model of the Alexa assistant can be specified into JSON-formatted data. Figure 2 shows an example of NLP to command a robot to turn on or off the light of the living room. In the left side, the red-colored text corresponds with the application intents. The only one intent has several

slots and utterances associated, which appear as green- and blue-colored text respectively. Each slot is of a defined data type, and they appear in figure as yellow-colored text. Note that, in the right side of the figure, each data type defined by the user is associated with a list of values it can take. The Amazon's assistant will only fire the intent *switchLight* when someone tells the echo device "tell him to turn on the living room light" or "tell him to turn off the living room light". Otherwise, no utterance will match with the given order and the voice assistant will do nothing. Figure 3 shows the Alexa assistant flow. It graphically draws the scheme described above.

```
"invocationName": "uma robot",
    "intents": [
      {
        "name": "switchLight",
        "slots": [
          {
            "name": "order",
            "type": "orderType"
          },
          {
            "name": "zone",
            "type": "zoneType"
          }
        ],
        "samples": [
          "tell him to turn {order} the {zone} light"
        ]
      }
    ],
```

```
"types": [
      {
        "name": "orderType",
        "values": [
            {"name": {"value": "on"}},
            {"name": {"value": "off"}}
        ]
      },
      {
        "name": "zoneType",
        "values": [
            {"name": {"value": "living room"}}
        ]
      }
  ]
```

Fig. 2: Alexa NLP JSON model example

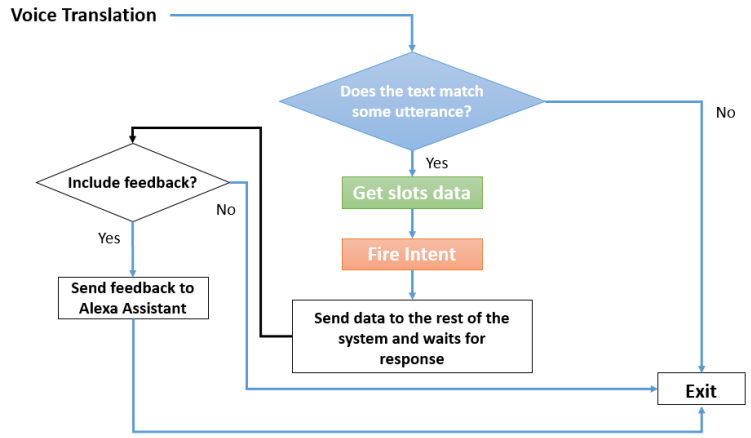

Fig. 3: Alexa NLP flow diagram

4 Tools, frameworks, and services

To program scalable voice-based software for robotics platforms there are several open-source tools, frameworks, and services available. One of the main advantages of these resources is that they can be self-hosted so, if their source code or executable are removed from their repositories, our applications can continue running. Next subsections present the resources that our proposal uses.

4.1 Docker

Docker[9] is an open-source project to develop and deliver software in packages called *containers*. Containers provide a layer of abstraction and automation of virtualization of applications that can run on different operating systems (OS). Docker also provides a mechanism for resource isolation, allowing that independent containers can run over a single instance of an OS. Summarizing, Docker containers are something like lightweight virtual machines (see Figure 4). In our proposal, Docker containers (in its version 18.06.1-ce) are used to deploy the whole system in an easy way. Thus, each self-hosted service, tool or framework is deployed into an independent Docker container.

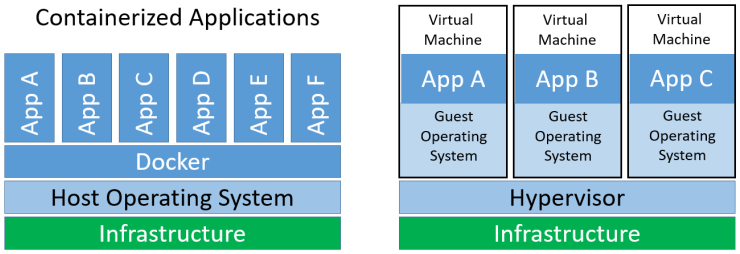

Fig. 4: Docker vs. Virtual Machines

4.2 Serveo

Alexa applications require an end-point URL to send user commands outside the Amazon's voice service server. This end-point is, in this case, the entry-point of the proposal we present in this paper. Amazon's voice service is designed to communicate over secure connections, mainly using the https protocol. However, in our case, the entry-point of our system is of type http and not of type https[10]. Hence, http queries need to be encapsulated into https queries, providing a compatible end-point to the Alexa applications. This process is known as ***tunneling***.

[9] https://docs.docker.com/engine/docker-overview/

[10] We do not have valid signed-certificates for the Alexa SDK and, furthermore, the chosen communication protocol, http or https, do not affect to the designed flow

Serveo[11] is a SSH-based service for port forwarding. It generates a valid https URL that anybody can use to connect with a specific remote server or service. In order to get this https URL, it is necessary to execute the following command in the server side (see Command 1).

Command 1: Obtaining an https URL using Serveo

```
$ autossh -M 0 -o "ServerAliveInterval␣120" -o "
    ↪ ServerAliveCountMax␣3" -R <custom_url>:80:localhost:<
    ↪ own_service_port> serveo.net
```

In Command 1, *autossh* is a Linux command that automatically restarts SSH sessions and tunnels. The autossh command is configured by setting several parameters. Among the ssh parameters, we found the *ServerAliveInterval* and *ServerAliveCountMax*. The first one indicates the number of seconds that the client will wait before sending a null packet to the server (to keep the connection alive). The second one indicates how long a client is allowed to stay unresponsive before being disconnected. Also, the -R option tells your SSH client to request port forwarding from the server and proxy requests to the specified host and port (usually localhost).

4.3 Node-RED and RedBot

Node-RED[12] is a very popular programming tool for wiring together hardware devices, APIs and online services as interconnected flows. It provides a browser-based editor that eases programming new application flows. Node-RED provides functionality through artifacts called ***nodes***. Nodes are configurable artifacts providing a specific functionality. For instance, we can define nodes implementing REST for rapidly deployment online services. A node output can be connected to one or several nodes input. In short, different nodes can interconnect with each other composing ***flows***. Moreover, Node-RED supports custom functionality by writing JavaScript code. This feature makes Node-RED a powerful application programming tool. Finally, a flow (or a set of them) models, implements and executes a whole system. Figure 5 shows two http REST services that have been modeled and developed using Node-RED. In our proposal, Node-RED (in its version v0.19.5) is mainly used to process data from Alexa assistant but also to model the interactions among all the components belonging to the system.

RedBot[13] (in our proposal using version 0.16.8) is an open-source Node-RED-based chat bot platform. This platform allows building full chat bots for Telegram, Facebook, Alexa, etc. through the previously described Node-RED's nodes. With respect to Alexa, RedBot provides, among others, nodes that act as entry-points for the Alexa queries; nodes that filter Alexa intents; nodes for inserting the user feedback as plain text (that the Alexa engine will later translate

[11] https://serveo.net/
[12] https://nodered.org/
[13] red-bot.io/

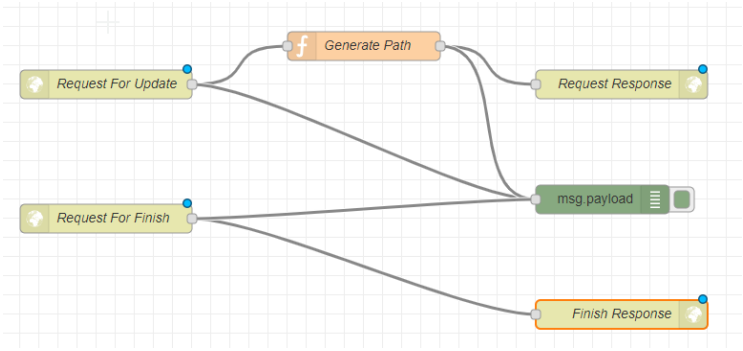

Fig. 5: Two http REST services developed as a Node-RED flow

into natural language using its NLG engine); and nodes for sending back the final response to the user query.

Although these Alexa-related nodes are the most important ones for the presented proposal, RedBot provides interesting functionality in the field of NLP. For example, programmers can use RedBot to develop custom chat bots (using RiveScript[14], Recast.ai[15], Dialogflow[16] or other NLP algorithms) and to easily extract relevant information (like numbers, email, etc.) from user's sentences.

4.4 Mosquitto

To explain what Mosquitto is, it is necessary to start explaining what MQTT is. MQTT (Message Queue Telemetry Transport) is a lightweight machine-to-machine (M2M)/IoT communication protocol that is based in a publish-subscribe mechanism. The MQTT publish-subscribe mechanism manages messages tagged under a plain name called ***topic***. In short, in a MQTT-based system, some software entities will send messages out using custom topics. Those messages will later be received by all software entities that are subscribed to the same topics. But in order to perform communication between two remote software entities, MQTT needs a ***broker*** (see Figure 6).

A MQTT broker is the element in charge of managing the network and transmitting messages. The most popular one is the Eclipse Mosquitto (simply called Mosquitto). Mosquitto is an open-source lightweight message broker that is suitable for use on all devices, from low-power single board computers to full servers. It is fully configurable and has plenty of features that allow programmers, for instance, to develop secure communications. Both MQTT and Mosquitto are designed to run over low bandwidth and high latency networks. Moreover, they require low computation capacity so they are one of the best choices when working with LE devices. For these reasons, they have becoming one of the

[14] https://www.rivescript.com/
[15] https://cai.tools.sap/
[16] https://dialogflow.com/

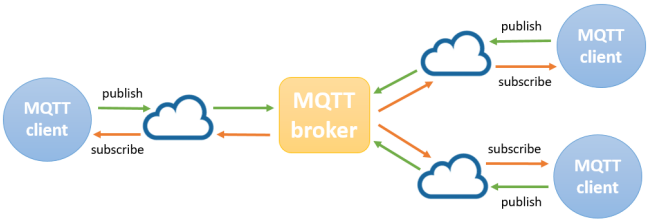

Fig. 6: The MQTT star topology

most popular combinations in the IoT field. In our proposal, they are used to communicate the external agents (distributed robots, in our application scenario) among them, and with the Node-RED server.

4.5 ROS

ROS[17] (Robot Operating System) is an open-source, meta-operating system to program robots. It provides OS features, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management among others. It also provides a set of tools and libraries to obtain, build, write, and run code across multiple computers. ROS models systems as a *runtime graph* where processes are in a peer-to-peer network. The processes in the runtime graph are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style (Remote Procedure Call) communication over services, asynchronous RTPS (Real-Time Publish-Subscribe) of data over topics, and storage of data on a Parameter Server.

In our proposal, ROS (using the kinetic version) is used in the robot side. Its objective is to model the robot system and execute the necessary logic to carry out the commands received from the user.

5 Embedded systems as dedicated services for voice-based applications

In Section 4, we presented the open-source-based resources employed in the proposed system. All these tools only require a few computation resources and therefore they are suitable to be hosted in single board embedded computers. In our experiments, two different single board embedded computers have been tested: the Odroid xu4 and the Raspberry pi 3 model B+. Table 1 summarizes the specifications for both devices. As the table shows, the Odroid board is worse than the Raspberry one in terms of consumption, however it has a better performance. As the remote server (running Node-RED, RedBot, and the Mosquitto

[17] https://www.ros.org/

MQTT broker) is always connected to the electrical network, the power consumption is not a critical parameter at this point. Also, and due to the fact that the remote server executes a very important part of the system (the Node-RED flows interconnecting the systems entities), an extra processing capacity is desirable. For these reasons, the Odroid xu4 was the selected board to act as remote server. Robots, on the other hand, are limited by its batteries, so the power consumption is a critical parameter. This is the main reason why the Raspberry pi board was selected as a robot controller, because it has enough computation capacity to meet software requirements (mainly to execute ROS) and it presents lower energy consumption than the Odroid board.

6 Designed Node-RED flows

Several Node-RED flows were designed and developed to interconnect the Alexa assistant with different devices (robots or other actuators). In this section we present one of these Node-RED flows (see Figure 7). In many cases, these actuators are home-made devices, not officially supported by the Alexa ecosystem. So, this flow can be seen as a starting point for those researchers who want to control their devices using the Amazon's voice assistant as a hands-free interface. Figure 7 provides a numbered snapshot of our proposed architecture. Numbers correspond to the most relevant parts of the flow, that is detailed below:

1. **Alexa Receiver**: This is the node acting as the Alexa application's endpoint. It receives user queries as JSON-formatted documents.
2. **Is intent**: It connects each Alexa assistant's intents with a specific node output. In short, it acts as an intents selector capable of modifying the Node-RED flow.
3. **Take decisions and send data to external agents:** Once an intent is selected, a JavaScript-based function takes a decision depending on the JSON at the input. It extracts the payload and sends the slot variables via MQTT

Table 1: Main features of the Odroid xu4 and Raspberry pi 3 model B+

Feature	**Odroid xu4**	**Raspberri pi 3 model B+**
Processor	Octa ARM Cortex-A15 Quad 2Ghz Cortex-A7 Quad 1.3GHz CPUs	Cortex-A53 (ARMv8) 1.4GHz
Memory	2GB LPDDR3 RAM	1GB LPDDR2 SDRAM
Graphics	Mali-T628 MP6	Broadcom Videocore-IV
OS	Ubuntu 18.04	Raspbian
Consumption	up to 15 Watts	up to 5.1 Watts

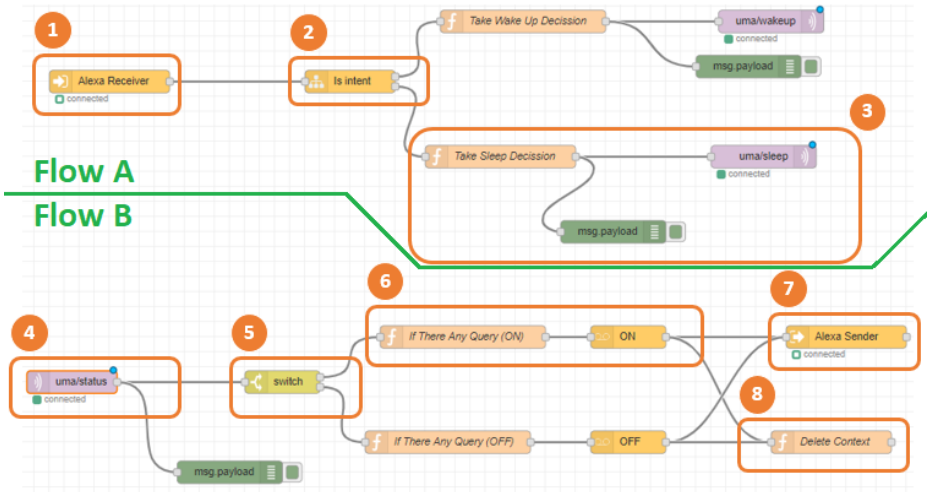

Fig. 7: Designed Node-RED flows (A and B) to interconnect (home-made) devices with the Alexa voice-based assistant

using a custom JSON message. Before the function finishes, it saves the original message into the *Node-RED context* to allow the rest of the nodes to use it later (if needed). Furthermore, a debugging node is used (green-colored node) to test functionality through the Node-RED's *Debugging Window*.

4. **MQTT subscription node**: This node receives an end-device's status via MQTT subscriber. After that, it sends the received response (the message payload) to the next stage in the flow.
5. **Switch node**: This node checks the message payload received from the previous stage, and informs whether the end-device is ready to perform any command sent by the user ('ON'), or not ('OFF').
6. **Alexa Speech and Query-Recognition**: A Javascript-based function checks if the current execution was caused by an user order through the Alexa assistant. This step is necessary as other non voice-based systems, which are also interacting with the robot, could have triggered this order. If the result is 'ON' or 'OFF', a speech is performed by the Alexa node reporting its status to the user. Note that the speech node uses the original message that was stored in context to carry out the NLG phase.
7. **Alexa Sender**: This is an output node to send back responses to the Alexa-based device. A queue with all given commands is set.
8. **Delete Context**: This function deletes messages in context to allow new queries to be executed correctly.

7 Conclusions and future work

Given the interest of large companies on introducing networked virtual assistants at home, it is expected that these devices will become popular in the next years.

These assistants are equipped with artificially intelligent frameworks, which allow users to interact with them and also to use them as interfaces for commanding other devices hands-free. This scenario offers the user the possibility to interface a smart environment using natural speech. In other words, it increases accessibility of smart environments, becoming a tool to increase autonomy for the most fragile people, such as the elderly or users with disabilities.

In this paper, we describe how the virtual assistant can be connected with other devices to allow users commanding them. There is no novelties in the contribution, but an ordered summary of all technologies involved in our proposed architecture is presented, and as described in the paper, the use of some of these tools is not straightforward. The paper pursuits to ease the programmers to create a VUI for controlling their home-made (or not) devices, using tools that have proven their robustness and flexibility (referred to programming) in everyday environments. This is the main contribution of this paper.

Future work focuses on extending this framework to a more ambitious scenario: interacting with elderly people in retirement houses. This application scenario is considered in the projects that are currently being addressed in our research group, and that continue the research topics of recently finished projects such as CLARC[18] or LifeBots[19]. In this scenario, a robust VUI is clearly required to increase the possibilities of a socially assistive robot to be accepted and used. Working with elderly people in retirement homes, and using the proposed VUI, our robots will be able to provide several services such as suggesting physical or cognitive activities, or showing information about the weather or menu. With the help of the virtual assistant and chat bot, we will also integrate the robots with other devices in the environment, increasing the repertoire of activities to be offered to the users, and allowing these robots become a familiar and welcomed presence in the shared environment of the retirement house.

References

1. Gartner's it glossary. https://www.gartner.com/it-glossary/virtual-assistant-va, accessed: 2019-06-15
2. Robotics 2020 multi-annual roadmap for robotics in europe. Tech. rep., SPARC: The partnership for robotics in Europe. The EU framework programme for research and innovation (2015)
3. Bandera, A., et al.: CLARC: a Robotic Architecture for Comprehensive Geriatric Assessment. In: Proc. XVII Workshop of Physical Agents (WAF 2016). Málaga, Spain (June 16-17 2016)
4. Chung, Y., Weng, W., Tong, S., Glass, J.: Towards unsupervised speech-to-text translation. CoRR abs/1811.01307 (2018)
5. Wachter, M., Ovchinnikova, E., Wittenbeck, V., Kaiser, P., Szedmak, S., Mustafa, W., Kraft, D., Krger, N., Piater, J., Asfour, T.: Integrating multi-purpose natural language understanding, robots memory, and symbolic planning for task execution in humanoid robots. Robotics and Autonomous Systems 99, 148–165 (2018)

[18] http://www.clarc-echord.eu/
[19] http://www.plg.inf.uc3m.es/ lifebots/