Cloud Services for Social Robots and Artificial Agents

Lucrezia Grassi[1], Carmine Tommaso Recchiuto[1] and Antonio Sgorbissa[1]

[1]*Laboratorium - DIBRIS, Università di Genova, via all'Opera Pia 13, 16145, Genova, Italy*

Abstract

This work presents the design and the implementation of CAIR: a cloud system for knowledge-based interaction devised for Social Robots and other conversational agents. The system is structured in a way that it can be easily expanded by adding new services that improve the capabilities of the clients connected to the Cloud. Another key feature of the system is that it has been designed to make the development of its clients straightforward: in this way, multiple devices (e.g., robots, computers, smartphones, etc.) can be easily endowed with the capability of autonomously interacting with the user, understanding when to perform specific actions, and exploiting all the information provided by services on the Cloud.

Keywords

Cloud Robotics, REST API, Client-Server Architecture, Socially Assistive Robots, Human-Robot Interaction

1. Introduction

CARESSES[1] is an international, multidisciplinary project whose goal is to design the first robots that can assist older people and adapt to the culture of the individual they are taking care of [1, 2]. The robots can help the users in many ways including reminding them to take their medication, encouraging them to keep active, helping them keep in touch with family and friends. Each action is performed with attention to the older person's customs, cultural practices and individual preferences. In CARESSES, the ability of the companion robot to naturally converse with the user has been achieved by creating a framework for cultural knowledge representation that relies on an Ontology [3, 4]. A major limitation of the CARESSES system is that only one device at a time can connect to the server and exploit all the capabilities provided by the system. Moreover, its server stores the information related to a single client, and it manages all the connections, even from different devices, as coming from the same user. This kind of implementation also prevents the system to allow for an expansion of the knowledge base from multiple users.

Recently, it has become more and more common to exploit cloud technologies to improve the efficiency of many devices and systems. In the robotics field, this practice is defined as *cloud*

The 8th Italian Workshop on Artificial Intelligence and Robotics – AIRO 2021

lucrezia.grassi@edu.unige.it (L. Grassi); carmine.recchiuto@dibris.unige.it (C. T. Recchiuto); antonio.sgorbissa@unige.it (A. Sgorbissa)

https://www.researchgate.net/profile/Lucrezia-Grassi (L. Grassi); https://www.researchgate.net/profile/Carmine-Recchiuto (C. T. Recchiuto); https://www.researchgate.net/profile/Antonio-Sgorbissa (A. Sgorbissa)

0000-0001-6363-3962 (L. Grassi); 0000-0001-9550-3740 (C. T. Recchiuto); 0000-0001-7789-4311 (A. Sgorbissa)

[1]www.caressesrobot.org

robotics, i.e. the use of remote computing resources to enable greater memory, computational power, collective learning and inter-connectivity for robotics applications [5, 6].

This paper describes the cloud system that has been designed based on the underlying principles of the CARESSES system, but completely modifying its architecture to offer a set of web services for multiple robots and devices.

The CAIR system has been developed by taking advantage of the rich knowledge base and the dialogue mechanism developed during the CARESSES project, intending to create a system much easier to use and able to manage contemporary connections from multiple users. The system is based on the use of REST APIs [7] that provide many advantages such as scalability, flexibility, portability and independence (see Section 2). CAIR web services allow the connected clients to manage a rich conversation with the users and to receive Plans to be executed, if possible, on the client device. Moreover, such architecture allows to effectively exploit the already implemented mechanism for knowledge expansion [8], which will be integrated into the following versions of the system.

The system for knowledge-based autonomous interaction described in this work can be easily used by most devices with Internet connectivity, able to acquire an input through a keyboard or a microphone, and provide an output through a screen or a speaker (e.g., robots, computers, smartphones, smartwatches, etc.).

2. System Architecture

The system is based on a client-server architecture and it has been designed in a way that it is easy to add new functionalities to improve its performance, and it is easy to be used by the clients.

Figure 1 depicts the proposed architecture. The server is composed of two web services developed in Python: (1) the Dialogue Manager service that manages the dialogue and analyzes the user sentence to recognize the intention of talking about a specific topic, and (2) the Plan Manager service that recognizes the intention of the user to make the agent execute a specific action. To provide appropriate answers and plans, the server exploits an Ontology containing all the topics, keywords, sentences and plans used during the interaction with the user. The Flask-RESTful[2] framework has been used when developing the web services.

The client can perform requests to the server using REST APIs. REST is a set of rules that should be followed when creating the API. One of these rules states that the client should be able to get a piece of data (called a *resource*) when linked to a specific URI. Each URI is called a *request*, while the data sent back to the user is called a *response*. Any web service that obeys the REST constraints is informally described as RESTful. Due to the separation between client and server, this protocol makes it easy for developments across the various areas of a project to take place independently. In addition, the REST API adapts at all times to the working syntax and platform. This offers the opportunity to try several environments while developing.

The client acquires the user sentence, sends it to the server, parses the response, executes the received Plan and/or replies with the dialogue sentence returned by the Dialogue Manager.

[2] https://flask-restful.readthedocs.io/en/latest/

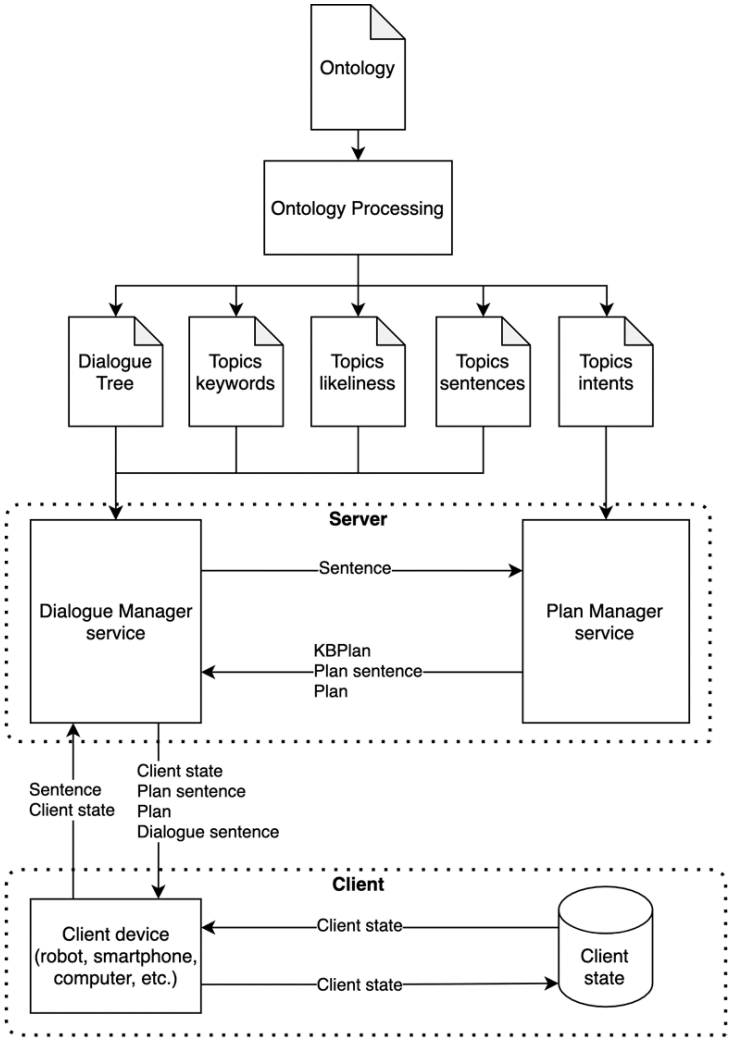

Figure 1: CAIR system architecture

2.1. Dialogue Manager service

The Dialogue Manager service is in charge of managing the dialogue. To provide the appropriate response to the user, the Dialogue Manager service requires as input the user sentence and the client state. As mentioned before, the ability of the system to naturally converse with the user has been achieved by creating a framework for cultural knowledge representation that relies on an Ontology [3, 4]. However, to deal with representations of the world that may vary across different cultures [9], the Ontology is organized into three layers, as explained more in detail in [10, 4]. The Ontology structure is used to build the Dialogue Tree (DT) and some additional files, namely the Topics keywords, Topics likeliness and Topics sentences, which are ultimately fed to the conversation system to chit-chat with the user.

Based on the Dialogue Tree, the key ideas for knowledge-driven conversation can be briefly

summarized as follows (the whole process [10] is more complex).

Each time a user sentence is acquired:

1. A Dialogue Management algorithm (either keyword-based or based on more advanced topic classification techniques) is applied to check if the user's sentence may trigger one of the topics in the DT by jumping to the corresponding node;
2. If no topics are triggered, the conversation follows one of the branches of the DT (according to policies that take into account the user's cultural background and personal preferences).

The system continues in this way, proposing sentences corresponding to a node and acquiring the user's feedback that can be used to update the user's preferences and/or determine the next node to move to.

2.2. Plan Manager Service

As shown in Figure 1, other than providing the dialogue reply and the updated client state, the Dialogue Manager service also calls the Plan Manager service, working as an intermediary between the client and such service. However, the client could directly call this service if not interested in managing the dialogue.

The Plan Manager service receives as input the user sentence. Its purpose is to find a match between such sentence and one of the trigger sentences of a specific Intent. An Intent is defined by (i) a set of trigger sentences (built using regexes), (ii) one or more plan-specific sentences (if any), (iii) a KBPlan (if any), (iv) and a Plan (if any) (Figure 2).

Trigger sentences (i) are used to check if the user sentence matches with the corresponding Intent. Such sentences are currently modeled with regexes, and allow to extract parameters from the matched sentence that can be used to dynamically compose the plan sentences, the plan and the KBPlan. Besides trigger sentences, also plan-specific sentences have been defined (ii), which are used by the system to reply when an intent is detected.

A KBPlan, where KB stands for Knowledge Base, is a sequence of actions meant to affect the knowledge base and/or the flow of the dialogue. For instance, if the user says *"I love music"*, this sentence will match the trigger sentences of the Appreciation Intent (Figure 2) meant to recognize the user's appreciation for something and extract the loved thing as a parameter. The KBPlan of this Intent is composed of two actions: the first one increases the probability that the user wants to talk about the extracted parameter, while the second one brings the information that the system should jump to that conversation topic (if present in the Ontology).

A Plan is a sequence of actions that should be executed on the client as it does not affect the knowledge base nor the flow of the dialogue. For instance, if the user says *"Play the song Yesterday"*, this sentence will match one of the trigger sentences of the Music Intent (Figure 2) that recognizes the user intention to some music. The plan of this Intent is composed of a single action carrying the information that the client should play the song having the title contained in the parameter field (see Section 2.3).

If the Plan Manager service finds a match with an Intent, a response containing the *KBPlan*, the *plan sentence*, and the *plan* is returned to the Dialogue Manager service (see Figure 1). The *KBPlan* is managed by the Dialogue Manager, as it directly affects the continuation of the dialogue, while the *plan sentence* and the *plan* are not used by this service. Once the Dialogue

Music Intent

1. **Trigger Sentences:**
   - (?:P|p)lay (?:some |a)?(music|song)_0
   - I want to listen (?:to)?(.*)_1
   - (?:P|p)lay (?:the)?song (.*)_1
2. **Plan sentences:**
   - Ok.
   - Sure.
3. **Plan**: [{'action':playsong, 'title': $parameter1}]
4. **KBPlan:** No KBPlan

Figure 2: Examples of Intent recognized by the CAIR system

Manager service has chosen the most appropriate answer for the user based on the information contained in the client state, the previous user sentence, and, eventually, the KBPlan, it returns a response to the client. Such a response contains the updated client state, the plan sentence, the plan, and the dialogue sentence (i.e., the actual continuation of the dialogue).

2.3. Client

A client for CAIR can be easily developed for most devices with Internet connectivity such as robots, computers, smartphones, smartwatches, etc. The first thing that the client should do is to perform a PUT request to the server, in particular to the Dialogue Manager service, to obtain the initial client state and the first sentence of the conversation. The client state should be stored locally and retrieved before all the following requests. Afterwards, the client should acquire the user input, send it to the server along with the client state with a GET request, retrieve and manage the response. This last operation expects the system to communicate the Intent reply to the user, perform the actions contained in the `plan` field of the response, and eventually continue the dialogue by communicating the sentence reply. If the client is not able to execute the actions it can ignore them and consider only the sentence reply.

An example of a simple client for PC can be found in this GitHub repository[3]. The repository contains also a PDF guide with a detailed explanation of the code and of all Plans that the server can return to the client based on the Intent that has been matched. Another documented example of a full client for the SoftBank Robotics robots Pepper and Nao, that manages all the Plans returned by the server, can be found here[4]. A video showing some extracts of the interaction is available on YouTube[5].

[3] https://github.com/lucregrassi/CAIRclient_instructions.git
[4] https://github.com/lucregrassi/CAIRclient/tree/develop
[5] https://www.youtube.com/watch?v=UdbYGytd07w

References

[1] C. Papadopoulos, N. Castro, A. Nigath, R. Davidson, N. Faulkes, R. Menicatti, A. A. Khaliq, C. Recchiuto, L. Battistuzzi, G. Randhawa, et al., The caresses randomised controlled trial: Exploring the health-related impact of culturally competent artificial intelligence embedded into socially assistive robots and tested in older adult care homes, International Journal of Social Robotics (2021) 1–12.

[2] A. Khaliq, U. Kockemann, F. Pecora, A. Saffiotti, B. Bruno, C. Recchiuto, A. Sgorbissa, H.-D. Bui, N. Chong, Culturally aware planning and execution of robot actions, 2018, pp. 326–332.

[3] N. Guarino, Formal ontology and information systems, FOIS'98 Conf (1998) 81–97.

[4] B. Bruno, C. T. Recchiuto, I. Papadopoulos, A. Saffiotti, C. Koulouglioti, R. Menicatti, F. Mastrogiovanni, R. Zaccaria, A. Sgorbissa, Knowledge representation for culturally competent personal robots: requirements, design principles, implementation, and assessment, International Journal of Social Robotics 11 (2019) 515–538.

[5] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, A. V. Vasilakos, Cloud robotics: Current status and open issues, IEEE Access 4 (2016) 2797–2807.

[6] C. Recchuto, L. Gava, L. Grassi, A. Grillo, M. Lagomarsino, D. Lanza, Z. Liu, C. Papadopoulos, I. Papadopoulos, A. Scalmato, et al., Cloud services for culture aware conversation: Socially assistive robots and virtual assistants, in: 2020 17th International Conference on Ubiquitous Robots (UR), IEEE, 2020, pp. 270–277.

[7] M. Masse, Rest api design rulebook: Designing consistent restful web service interfaces, O'Reilly (2011).

[8] L. Grassi, C. Recchiuto, A. Sgorbissa, Knowledge triggering, extraction and storage via human–robot verbal interaction, Robotics and Autonomous Systems 148 (2022).

[9] M. Carrithers, M. Candea, K. Sykes, M. Holbraad, S. Venkatesan, Ontology is just another word for culture: Motion tabled at the 2008 meeting of the group for debates in anthropological theory, university of manchester, Critique of anthropology 30 (2010) 152–200.

[10] C. T. Recchiuto, A. Sgorbissa, A feasibility study of culture-aware cloud services for conversational robots, IEEE Robotics and Automation Letters 5 (2020) 6559–6566.