Cozmo4Resto: A Practical AI Application for Human-Robot Interaction

Kevin El Haddad [(1)], Noé Tits[(1)], Ella Velner [(2)], Hugo Bohy[(1)]
[(1)] Numediart Institute, University of Mons, Mons, Belgium
[(2)] commercom, Amsterdam, The Netherlands
kevin.elhaddad@umons.ac.be, ellavelner@gmail.com, noe.tits@umons.ac.be,
hugo.bohy@student.umons.ac.be

***Abstract*—In this paper we report our first attempt on building a Human-Agent Interaction (HAI) open-source toolkit to build HAI applications. We present a human-robot interaction application using the Cozmo robot built using different modules. The scenario of this application involves getting the agent's attention by calling its name (Cozmo), then interacting with it by asking it for information concerning restaurant (e.g: "give me the nearest vegetarian restaurant"). We detail the implementation and evaluation of each module and indicate the future steps towards building the full open-source toolkit.**

***Index Terms*—Human-Agent Interaction (HAI), Human-Robot Interaction, deep learning, Text-to-Speech Synthesis (TTS), Keyword Spotting, Automatic Speech Recognition (ASR), Dialog Management, Sound Localization, Signal Processing, Cozmo.**

I. Introduction

THE past decades witnessed the rise of Human-Agent Interaction (HAI) systems such as conversational agents and intelligent assistants. This work aims at contributing to the improvement of HAI applications and their incorporation to our daily lives. HAI systems are generally formed of different modules with different task(s) each, communicating with each other.

We aim at building a toolkit containing such modules, as well as a framework with two main purposes:

1) controlling the agent's behavior in a user-defined way;
2) connecting these modules together in a single application so that they could be able to communicate with each other in a user defined logic;

The goal is to have a toolkit allowing the users the most freedom possible in the way they utilize it to build their HAI applications. The above mentioned modules would thus be usable either in an "off-the-shelf" mode (outside the framework) or in the framework defined here.

In the same perspective, in the future, modules will be incrementally added to this toolkit allowing a wider range of HAI applications implementations. Also, the framework is designed in a way to easily add and connect modules needed (toolkit's ones or user defined ones) in order to build customized HAI applications. This gives users more freedom on how to utilize the toolkit.

In order to evaluate the performance of the developed toolkit in building HAI systems, an application will be developed using it: Cozmo4Resto. This HAI application is an interaction with the Cozmo robot [1] during which Cozmo will give the user informations about restaurants based on the user's queries as described in further detail in Section III. This robot was chosen mainly because of the simplicity of integration in a python-based application (see also Section III).

Towards building this application, in this paper, we present the modules developed to be used for Cozmo4Resto and added to the toolkit, as well as the framework mentioned above. We will therefore first present the HAI-toolkit in general in Section II. Then detail the Cozmo4Resto application is explained further and the modules developed detailed in Section III. Finally the implementation of the platform for Cozmo4Resto is detailed in Section IV.

II. HAI Open-Source Toolkit

We present here the first version of this toolkit [2] that will be used to implement HAI application like Cozmo4Resto. It is implemented in a modular way and, as mentioned earlier, can be viewed either as a framework upon which modules are connected and the agent's behavior is controlled to build an HAI application or as a library of HAI-oriented modules usable outside the framework.

A. Modules

A module's task is to perform an action or a sequence of actions which is/are part of the agent's behavior and which is/are needed in the application implemented. The input-output of each module is implemented in an object oriented way and will have a specific and fixed format. This way, each module can be modified/replaced/improved without affecting the implementation of the others. This will help making the toolkit more generic.

B. Behavior Framework

The framework's main purpose is to allow the integration of all the different modules in a single HAI system. It can be summarized as a finite state machine [1] (FSM)-based system combined with a communication system.

The FSM is used to describe the agent's behavior. Each state corresponds to a specific behavior of the agent. In the

[1] https://anki.com/en-us/cozmo.html
[2] https://github.com/kelhad00/hai-toolkit

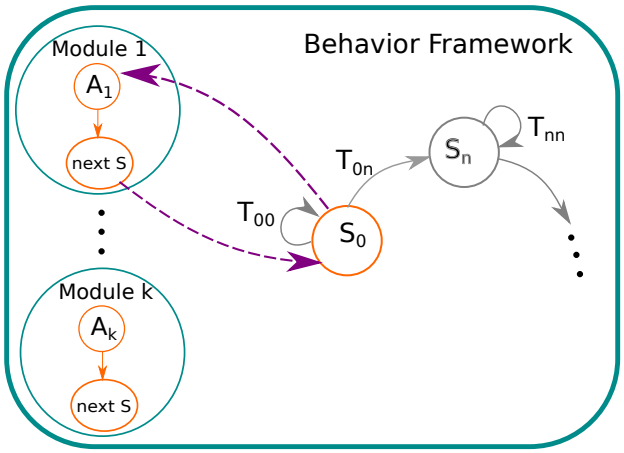

Fig. 1. Behavior framework workflow. S being the state, T the transition between states, A the action performed while in the state corresponding to the Module containing the action.

FSM, each state is linked to a module. Each module contains a sequence of actions the agent must perform while in the corresponding state.

The communication system allows the exchange of messages/data between the different modules. The main benefit of this is modularity and the ability to run each module on the same machine or different ones.

In what follows, we will refer to this framework as the "Behavior" framework, because it is used to describe the behavior of the agent through the states and the transitions between them. A visualization of the platform's workflow can be seen in Fig. 1.

Let $S = \{s_0, s_1, ..., s_n\}$ be the set of states describing the agent's behavior (s_0 being the initial state), T_{ij} the transition from state i to j and A_k is the action performed by a module to which the state can be connected.

During run time, the Behavior framework acts as a client to each of the modules, which therefore act as servers. When in state s_i, *Behavior* queries the module to which s_i is linked with the input required. After the action is executed, a "next state" value is returned triggering the transition to another state or to the same current state.

The link between states and modules is defined by the user.

III. Cozmo4Resto Application

As mentioned above, to validate our toolkit, we use it to build an HAI application involving the Cozmo Robot.

Cozmo [3] is a small physical relatively cheap robot which is designed to interact with users in games and other kinds of user-defined modalities. Cozmo can be very expressive through the eyes, audio and movements (body and head). A python SDK [4] is provided with Cozmo for free, which makes it easy to intergrate on several platforms and with different applications. The SDK python commands are sent to Cozmo via and android-based app that was developed for Cozmo. It contains a camera of which the stream is accessible via the SDK as well as other sensors. All this makes Cozmo an ideal platform to test our toolkit.

The interaction scenario of this application can be described as follows:

1) Cozmo would be wandering in a "non-interactive" mode;
2) when the keyword "Cozmo" is detected, Cozmo will turn around toward the caller and engage the interaction, thus going into "interactive" mode;
3) the user will query Cozmo concerning different information about restaurants (opening hours, menus, proximity, etc.);
4) after the interaction ends, Cozmo goes back to the "non-interactive" mode.

For this, three modules are needed: sound acquisition, keyword spotting (KWS) and sound localization (LOCAL). The sound acquisition module stores audio data in an efficient way for it to be used later on. The KWS detects a specific sound among others and LOCAL detects the directionality of the sound's source allowing to make Cozmo turn towards it. These will trigger the "interactive" mode. During the interaction an Automatic Speech Recognition (ASR) system will be used to convert the user's speech signal to text, which will be sent to a Dialog Management (DM) module. The DM module will take care of understanding the utterance and generating a text response based on an implemented logic (see Section III-Efor a more detailed description of the dialog interaction). The response will be sent to a Text-To-Speech (TTS) synthesis system which will take care of converting the text response into an audio speech signal.

In what follows we will explain our approach to building and/or testing each module (some modules were already implemented open-source systems). The main constraints being the quality of the system and the computation time. Indeed a trade-off needs to be found so that the entire system generates high quality responses in a reasonable delay of time. We will also present the framework mentioned in SectionII. The modules described in the following will be incorporated in the HAI-toolkit in general and are not meant only for the Cozmo4Resto application.

A. Sound Acquisition

The python pyaudio library [5] is used to acquire the audio. A ring buffer is used to store and stream the input sound. The recording starts when the signal reaches a certain threshold The recording stops when the signal goes below the threshold The buffer is created by using the deque function from the collections python library. Each shift of audio signals recorded by the stream is added to the buffer, until it reached its maximum length. This maximum length is passed as a parameter at the creation of the buffer. Once the buffer is full, every new shift overwrites oldest one in the buffer. A number of channel (one per mic input) can be specified.

[3]Please note that at the time of writting this report, Anki, the company producing Cozmo, went bankrupted. But the SDK is still maintained at the time of redaction.

[4]https://developer.anki.com/

[5]https://pypi.org/project/PyAudio/

B. Sound Localization

Theory: The goal is to find the direction of arrival of a sound to a microphone or a set of microphones. For this we use the time delay of sound reception between the microphones. Three microphone positioned as shown in Fig. 2 and 3 are used here. The exact time of the sound emission from the source is unknown, so the time delays between reception time at each microphone are used instead. In Fig. 2, 'Source' is the sound's source and 'T_{mic_i}' is the absolute time of reception of the microphone i. The source's coordinates (x, y) are determined by minimizing the both following equations using the root() function from scipy.optimize python library :

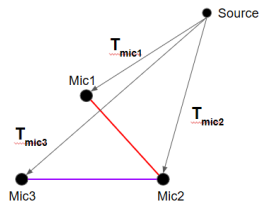

Fig. 2. Direction of arrival principle

$$v \cdot \tau_1 = \sqrt{(x_2 - x)^2 + (y_2 - y)^2} - \sqrt{(x_1 - x)^2 + (y_1 - y)^2}$$

$$v \cdot \tau_2 = \sqrt{(x_3 - x)^2 + (y_3 - y)^2} - \sqrt{(x_2 - x)^2 + (y_2 - y)^2}$$

Where (x_i, y_i) are the microphone i coordinates and τ_i is the delay between $T_{mic_{i+1}}$ and T_{mic_i}

Technical setup: Two different hardware setups are considered for this module. The first one is composed of 3 AmazonBasics Microphones disposed in an equilateral triangular shape of side one meter (see Fig. 3). The second setup is composed of one Raspberry Pi 3 Model B and a 6-Mic Circular Array Kit from Seeed's Respeaker (see Fig. 4). Only 3 of the 6 microphones are used in the later. The red circles in Fig, 4 shows an example of the relative positions of the mics used among the 6 available for this setup.

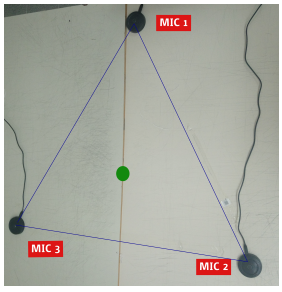

Fig. 3. Table Microphones setup

Database collection: In order to evaluate the above mentioned algorithm, we collected a dataset of different sounds with the different setups mentioned above. The sounds are either hand clapping or the word 'Cozmo', at different distances and angles with respect to the microphones: the angles vary by 30 degrees from 0 to 330 degrees and the distances are approximately 2m and 1m. The sounds were recorded in 2 different conditions:

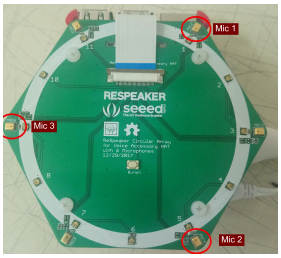

Fig. 4. Seeed's 6-Mic Array

- UMONS: recordings made in a relatively echo/reverberation-free room at 2m of the center of the microphones at the numediart institute of the University of Mons[6].
- ENT: recordings made in a room generating echo/reverberation at distances $>= 2m$ and $< 2m$ at Bilkent University[7].

The algorithm described above was evaluated on the ENT condition only (data with reverberation) by calculating the cosine similarity between the angle estimated and the corresponding real value. The mean cosine similarity of all angles are shown in Table I in degrees, per sound and distance. This table shows the error between the actual position of the sound source and the estimated one.

TABLE I
MEAN COSINE SIMILARITY. COZ = COZMO, FAR=2M, CLOSE=1M, SAME=AT TABLE HEIGHT, HIGH= 80CM HIGHER THAN TABLE HEIGHT

clap-far	clap-close	coz-far-high	coz-far-same	coz-close-same
26.19	10.89	6.77	6.04	2.82

The results indicate that the closer the source is from the microphones, the better is the estimation of the angle. These errors suggest that the type of sound might affect the efficiency of the sound localization algorithm used here. This is probably due to the difference in sound parameters like the sound amplitude and also the reverberation/echo generated by each sound.

Also, the effect this error might have on the user experience and user perception during an HAI application is an interesting and important aspect to consider.

Both of points will be investigated in future work.

C. Keyword Spotting (KWS)

As mentioned previously, the role of the KWS in this project is to trigger the "interactive" mode. In our case we use the keyword "Cozmo". A small dataset of "Cozmo" utterances from different speakers and in different tones was collected for the purpose of this work.

A benchmark of KWS systems is available comparing 3 systems online [8]: Picovoice, Snowboy and PocketSphinx. This benchmark uses crowd-sourced words to train and evaluate

[6]https://numediart.org/
[7]https://w3.bilkent.edu.tr/bilkent/
[8]https://github.com/Picovoice/wakeword-benchmark

TABLE II
AVERAGE WORD ERROR RATE (WER) AND DURATION OF COMPUTATION OF SENTENCES OF IEMOCAP DATASET

	Google Speech Recognition	DeepSpeech	Sphinx
WER	0.30	0.38	0.55
duration	1.69	0.8	9.5

these systems. The data used, therefore comes from different recording environments.

The customisation of Picovoice is done with text data. It relies on a dictionary of words with their corresponding phonetics. This dictionary is not accessible, and the word "Cozmo" needed for our application is not included in it. It is therefore not adequate for our application.

PocketSphinx is a mobile device version of Sphinx that runs locally, a group of speech recognition systems developed by Carnegie Mellon University. It uses HMMs for statistical modeling and includes a keyword spotting module. Similarly to Picovoice, it relies on a dictionary of words with phonetics.

Snowboy uses an API to send trigger words samples to train a system which is then downloaded and run locally. No more than three samples can be used to train the models.

Snowboy seems therefore like the best option for our application. We will therefore use "Cozmo" as training sample.

It is important to note that neither Snowboy or Picovoice systems are fully open source. Indeed the model training of Snowboy is performed through their web interface and Picovoice is optimized with a binary files provided online.

D. Automatic Speech Recognition (ASR)

Several ASR APIs are available for use under certain conditions. Some of these APIs are free but come with limitations of use in terms of API calls. Using APIs means we are dependent on a tier service that may not be free or not supported in the future.

A benchmark of APIs was proposed in [2]. Their code is open source[9].

Research projects with open-source codes include Sphinx, DeepSpeech, Kaldi toolkit and gentle (based on Kaldi toolkit).

In this section, we compare APIs and State-of-the-art open source systems for our use-case. For this, we estimate their performance in terms of the Word Error Rate based on the Levenshtein distance. We use the IEMOCAP database to approach a setup closer to our use-case of interaction compared to a database based on Audiobooks recordings used in [2].

DeepSpeech may be accelerated with GPU. This was used on Google Colab with a Tesla K80 GPU.

Table II reports the average WER and duration for obtaining the prediction of the sentences of IEMOCAP dataset.

E. Dialog Management (DM)

To manage the dialogue between the agent and the user, a module needed to be implemented to extract the goal of the user's utterance and to give the right response back. This is called a dialog management system. There are several options to create a working dialog management system, like using DialogFlow (Google) or Luis (Microsoft). However, we wanted an open-source library that worked with Python and JSON. Therefore, we chose the Rasa Library [10].

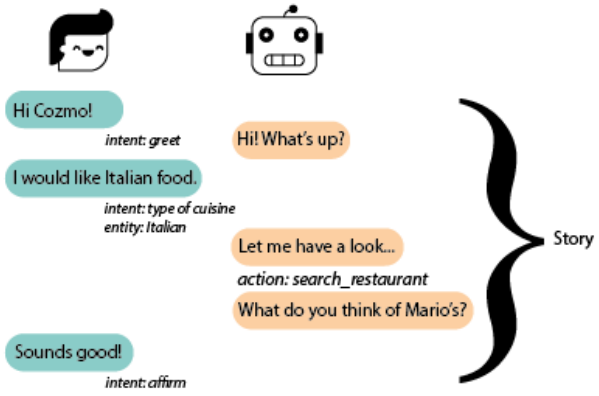

Fig. 5. Chat example with Rasa system.

RASA: This library is made up of two parts: Rasa NLU and Rasa Core. Rasa NLU extracts so-called 'intents' from the user's utterance. These intents are what the user wants or needs. These can be narrowed down by specifying 'entities'. For example, when the user says: "I would like Italian food", the intent is *the type of cuisine* and the entity is *Italian*. These intents and entities are then specified in a domain file, together with the template sentences that the agent can use to respond. This is where Rasa Core comes in. Rasa Core takes care of what the agent should do next, which most of the time is saying something back. But besides just responding, the agent can also perform actions, which are specified in a separate python file. For example, when the user asks for a restaurant nearby, the action search_restaurant is called, and it uses an API to extract restaurant details, to then give these to the user. An example of the start of a conversation is shown in Fig. 5.

The conversation altogether is a 'story'. Stories are predefined storylines the developer can create. These stories are the training data for Rasa Core. The training data for Rasa NLU are the intents. They are both trained using a neural network in Keras, based on an LSTM. This can be adjusted if necessary. The training data was created manually and is made available with the toolkit.

Cozmo4Resto & Rasa: Since the goal of Cozmo4Resto is to suggest restaurants nearby the user, the main action of Cozmo was to get the coordinates of the user (via their profile), find restaurants nearby of the type the user wants, and, if required, also give the address of the restaurant of choice. For the sake of simplicity, and for this application, we set the user location to a fix value and we propose only a single suggestion. Therefore we had two actions: action_search_restaurant and action_give_address. However, in a conversation people don't immediately ask what they want. The conversation usually has an introduction first (most often some sort of greeting). After

[9]https://github.com/Franck-Dernoncourt/ASR_benchmark

[10]https://rasa.com/docs/rasa/

TABLE III
THE CHANGES MADE DURING TESTING.

round	change
1	added stories
2	if cuisine not recognised: ask another
3	added action_other_suggestion
4	if cuisine is 'anything': give random restaurant
5	adjusted fallback method
6	added entities

TABLE IV
MEAN OPINION SCORE OF THREE PARTICIPANTS OF SENTENCES SYNTHESIZED WITH DIFFERENT TTS SYSTEMS

MOS	IBM API	gTTS	SOTA batched	SOTA unbatched
P1	2.80 ± 0.10	3.24 ± 0.16	3.70 ± 0.16	3.88 ± 0.13
P2	3.43 ± 0.23	3.28 ± 0.25	3.10 ± 0.36	3.63 ± 0.26
P3	2.25 ± 0.40	2.30 ± 0.31	2.10 ± 0.37	2.90 ± 0.43
All	2.83 ± 0.19	2.94 ± 0.18	2.97 ± 0.24	3.47 ± 0.19

this, the robot can ask the user a question leading to the goal of the conversation. When the goal is reached, the conversation comes to an end, with some kind of goodbye-utterance. This results in six intents: greet, goodbye, wantdinner, cuisine, affirm, deny. Wantdinner let's the agent know that the user wants a suggestion to eat somewhere. If there is no cuisine suggested already, the agent will ask for the type of restaurant. This will then lead to a 'cuisine'-response from the user. After receiving the type of cuisine, it is put in a Slot, so the actions can 'grab' this when needed. Cozmo will then look for a restaurant, with the help of the Zomato API, an API to extract information of restaurants [11]. Since we work with Python, the zomathon library was used [12]. When a restaurant with the right cuisine was found, Cozmo gave it back to the user, by saying 'What do you think of Abc, a restaurant that serves xyz?'. The user could then either 'affirm' or 'deny' this restaurant. If denied, the agent should give another suggestion, however, this is not implemented yet. When affirmed, the agent asks if the user needs the address, and if they affirm, the agent gives it, using the action_give_address. After this, the goal is reached, and the conversation will close with a goodbye (and a 'bon appetit!' from Cozmo).

Evaluation: To test if the dialog management system was working properly, nine eNTERFACE participants were asked to chat with the system in a simple command line interface. They were instructed to get information on a restaurant in the neighbourhood but were not informed about how to do this, to make the user's utterances as free and unguided as possible. When they felt the conversation was complete, or were stuck and could not go any further, they informed the researcher. The average conversation was about 4,5 minutes. Afterwards, they filled out an evaluative questionnaire, with five questions on conversational fluency from Mirnig et al. [3], put on a 7-point Likert scale, and an open question about what problems occurred. We iteratively improved our dialog strategy based on the results of the questionnaires and the conversation which was recorded in the system's logs. Therefore, after the first three conversations and then after each one, the system was improved and tested again. An example of an improvement is an adjustment to the fallback method ("Sorry, I did not understand that."). This went on until the participants did not seem to run into any problems. The utterances of the users also became new training examples for the system. The changes made after each round are shown in table III.

Implementing it in the toolkit: To have the DM working within the toolkit, it needed to be able to run outside the command line, and to be able to handle a JSON input file (from ASR), run the DM, and output a JSON file (to TTS). This was done by creating what is called a 'connector'-file. This contains the specifications on the input channel and a blueprint (from sanic [13]) on how to handle the input, namely how to send it to Rasa Core and retrieving Cozmo's responses. Since the toolkit was not entirely done by the end of eNTERFACE'19, the DM was made operable by connecting it to a Google Assistant.

F. Text-to-Speech (TTS)

As for ASR, some companies provide APIs for synthesizing speech from a text. Among them, *gTTS* is a python library allowing to use Google Translate built-in synthesizer. IBM provides the Watson TTS API[14].

One of the best state-of-the-art (SOTA) Open Source implementations in terms of naturalness so far for TTS is the joint implementation of Tacotron [4] and WaveRNN [5] systems in PyTorch[15]. Tacotron generates a mel-spectrogram from text and WaveRNN generates the corresponding waveform sample by sample from the predicted mel-spectrogram. WaveRNN is able to produce a very natural sounding audio wave but generating the signal sample by sample with a recurrent relationship is still slow. This implementation proposes a way to accelerate generation of a sentence, called *batched*, by generating segments of the signal output of a sentence in parallel. The segments have the to be concatenated together via a windowing process. This technique allows faster generation but leads to a chopped signal which is not the case of the *unbatched* generation.

For subjective evaluation, the 20 first sentences of harvard sentences[16] were synthesized. Then three people evaluated them subjectively in terms of naturalness by assigning a score between 1 and 5. The Mean Opinion Score was then computed for each system.

Table IV shows the results of the MOS test for each participant and each system.

The synthesis duration is also an important aspect to consider since this module will be integrated in a HAI application where the agent has to respond in real-time. Concerning the SOTA systems, the durations of generation of mel-spectrograms and waveforms using a GPU GeForce GTX 1080 Ti are summarized in Table V. The order of magnitude

[11] https://developers.zomato.com/api
[12] https://github.com/abhishtagatya/zomathon
[13] https://sanic.readthedocs.io/en/latest/sanic/blueprints.html
[14] https://www.ibm.com/watson/services/text-to-speech/
[15] https://github.com/fatchord/WaveRNN
[16] http://www.cs.columbia.edu/ hgs/audio/harvard.html

TABLE V
DURATION (IN SECONDS) OF GENERATION OF MEL-SPECTROGRAM (ABBREVIATED MEL) WITH TACOTRON AND WAVEFORM (ABBREVIATED WAV) WITH WAVERNN.

	mel	wav batched	wav unbatched
Mean	0.147419	12.771765	55.560221
Std	0.015291	0.173554	6.270252

for mel-spectrogram generation is 0.1 seconds. While for the waveform synthesis, it is one to several tens of seconds. In the unbatched mode, the standard deviation is much bigger because it depends more on the length of the sentence.

Therefore for a real-time interaction application one of the APIs is best suited since they both obtained reasonable and similar results on the MOS test and run relatively fast.

IV. COZMO4RESTO FRAMEWORK IMPLEMENTATION

For this application, the Behavior framework (client) will be connected with each module corresponding to the current state using the messaging library ZeroMQ [17] (for other application, the toolkit allows the use of other messaging systems). The input and output of each module are in the JSON format containing three values: the data needed by the module as input or returned by it as output, the current state and the next state.

The state machine describing Cozmo's behavior in the case of the Cozmo4Resto application is detailed Fig. 6. But, due to the eNTERFACE workshop's time constraints, the platform was evaluated using a simpler application which is described as follows:

- state listening: A user's speech is converted into text using an ASR
- state thinking: keywords are mapped to other words in a dictionary playing the role of a very simplified dialog management system.
- state speaking: the words from the dialog management are sent to a TTS system to be synthesized state listening: the system goes back to the ASR

The aforementioned is provided to the reader for testing [18].

V. CONCLUSION

In this paper, we report on the advancement in our project, presenting our goal for an open-source HAI toolkit and its application in our Cozmo4Resto project. In the future we will focus on integrating all the modules in a real-time application and test it in subjective experiments. Finally a generic library will be released as a first version of our open-source toolkit.

In future works, we will focus on implementing the behavior described in Fig. 6 by implementing the modules described previously into the platform. The whole system will then be tested in subjective evaluations by asking participants to first interact with Cozmo and then grading different aspects of the interaction like how well did the agent "understand" the requests made by the user, the accuracy of the responses, the delay between the questions and reactions.

We will use the toolkit to create other HAI applications with platforms other than Cozmo serving us as agents such as 3D avatars [6].

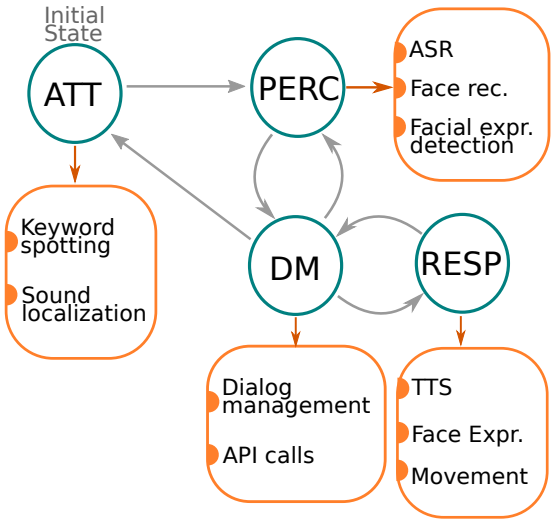

Fig. 6. State machine describing Cozmo's behavior for the Cozmo4Resto application. ATT: attention state which is linked to modules performing keyword spotting and sound localization-it is the initial state of the behavior, PERC: perception state which is linked to modules such as speech recognition (ASR) and face recognition, DM: the dialog management module which can also interact with API to harvest data from the web or control Cozmo directly, RESP: the response state generates a reaction to the user such as synthesized speech or a generated movement.

REFERENCES

[1] D. R. Wright, "Finite state machines," *Carolina State University*, p. 203, 2005.
[2] F. Dernoncourt, T. Bui, and W. Chang, "A framework for speech recognition benchmarking." in *Interspeech*, 2018, pp. 169–170.
[3] N. Mirnig, A. Weiss, G. Skantze, S. Al Moubayed, J. Gustafson, J. Beskow, B. Granström, and M. Tscheligi, "Face-to-face with a robot: What do we actually talk about?" *International Journal of Humanoid Robotics*, vol. 10, no. 01, p. 1350011, 2013.
[4] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. V. Le, Y. Agiomyrgiannakis, R. Clark, and R. A. Saurous, "Tacotron: Towards end-to-end speech synthesis," in *INTERSPEECH*, 2017.
[5] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. v. d. Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," *arXiv preprint arXiv:1802.08435*, 2018.
[6] K. El Haddad, F. Zajega, and T. Dutoit, "An open-source avatar for real-time human-agent interaction applications," in *Proceedings of 8th International Conference on Affective Computing and Intelligent Interaction*, 2019.

[17] https://zeromq.org/
[18] https://github.com/kelhad00/hai-toolkit