University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,
Computer Science and Engineering Department

BACHELOR THESIS

Human-Robot Interaction in a Social Assistive Context

Scientific Advisers:
Prof. dr. ing. Ţăpuş Nicolae
Prof. dr. ing. Ţăpuş Adriana

Author:
Ciocîran Ştefan-Dan

Bucharest, 2017

Abstract

The purpose of this thesis is to implement three new modules for the ENRICHME project, which is working with robots that interact with humans in a Social Assistive Context. Most of the humans are eldery people with mild cognitive impairment. The first two modules are graphical user interface modules: a **Social Module** which help the user to connect with his family trough internet, a **News Module** which agregates news from different newspapers. The Social Module offer a chat part and a video-conference part using the WebRTC tehnology. This thesis provides a new framework over WebRTC which can be used by developers who does not have a substantial knowledge in networking and WebRTC. The News Module makes use of the RSS feed and transform it in JSONP which can be used for secure connections and inside JavaScript scripts. On this news module the thesis provides an experiment that verify its adaptation. The third module (**Heart Rate Bluetooth Sensor Module**) is a ROS module that provides a heart rate sensor data over a bluetooth low energy connection into a ROS topic.

Contents

List of Figures

Notations and Abbreviations

BLE – Bluetooth Low Energy
HRBSM – Heart Rate Bluetooth Sensor Module
NM – News Module
SM – Social Module

Chapter 1

Introduction

This work is part of the Horizon2020 ENRICHME European project, whose purpose is to develop a socially assistive robot for elderly with mild cognitive impairment (MCI) to help them remain active and independent for longer and to enhance their quality of life. The project has partners from Italy, United Kingdom, France, Netherlands, Poland, and Greece. The system is going to be tested in Ambient Assisted Living (AAL) laboratories and in elderly housing. The AAL laboratories are Fondazione Don Carlo Gnocchi, in Milano, Italy, and Stichting Smart Homes in the Netherlands. In both facilities 15 older people will take part in the testing phase. The elderly caring facilities are Lace Housing Ltd, in the United Kingdom, Osrodek Pomocy Spolecznej in Poland, and Aktion Licenced Elderly Care Units in Greece. In each of the three sites 9 elderly will interact with the robot. The robot chosen for this project is the Tiago robot developed by PalRobotics. In each of the testing centers there is going to be one robot. The system is going to be simultaneously tested in all the testing sites. The people that can participate in testings need to have over 65 years old, they have to receive the diagnostic of MCI by a neuropsychologist or to be identfied with mild cognitive dementia on the MMSE score. Another aspect considered is the independence in executing basic activities for daily living assessed through Barthel ADL index. And lastly, the people need to be living in one of the studied facilities. "ENRICHME" refers to the goal of enriching the day-to-day experiences of elderly people at home by means of technologies that enable health monitoring, complementary care and social support, helping them to remain active and independent for longer and to enhance their quality of life. This work is focused on the time-extended personalized interaction while remaining in its own home environment. The only comparable technology is computer-based interaction, which could provide personalized care but without the physical embodiment of a robot. The experiments proposed in this work address the key differences between robot-based and computer-based interaction. [1]

1.1 Thesis description and objectifs

This thesis contains the description and implementation of the three modules inside the ENRICHME project. Two of them are graphical user interface module which are used in the web application presented on the Thiago robot (the robot of the project). The other one is ROS module which takes data from an hardware sensor (Heart Rate Sensor) and publish it on a ROS topic so it can be accesible by other modules and used.

For the first two the objectifs are:

- User interface that can be understand even if the user have some issues of seing.

- An intuitive and simple aplication that can be used first time without training.
- Portable Aplication - can be used on multiple robots.
- Adaptive Aplication - it adapts after the user.

For the ROS module the objectifs are:

- Real-time - the module must work in real-time.
- Interacting with other modules - it can interact with the other ROS modules.
- Error handling - it handles some errors unrealistic

Before the implementation of this modules the ENRICHME project did not have many interaction with things outside the AAL laboratories and elderly housing. The Social Module allow users to interact between them with chat messaging or video-conference calls. The News module takes news from internet sites and provides their content to the user. This modules is a step foward for introducing data outside the AAL and see their affects on the users. For the News Module the thesis contains an experiment on its adaptation and effects on humans. All the interface modules must be compatible in all the language used in the project.

1.2 ENRICHME project

Improving the quality of life of the elderly and the less able people has been for a long time one of the main challenges for the professionals in robotics. But in the last few years, technological progress and the development of common software platforms started to have a great positive impact in this field. The European project ENRICHME which aims to better monitor and prolong the independent living of the old people has developed an integrated system that can substantially help at achieving this goal. The project develops new technologies that enable health monitoring, complementary care and social support, helping elderly to remain active and independent for longer and to enhance their quality of life. More specific, the project tries to consolidate mobile robots within a smart-home environment o provide new Active and Assisted Living (AAL) services for the older person. Moreover, the project tries to combine ambient intelligence (smart home sensors and services), robot intelligence(HRI, robot sensors and services) and social intelligence (networked care, medical interface). ENRICHME system uses technological solutions and it mostly focuses on the integration of smart-home and robot sensors for human and object localisation, activity monitoring and detection of abnormal behaviours. The forth most important aspects of the project are:

1. Distributed architecture for Ambient Assisted Living (AAL) services based on ROS2;
2. Probabilistic solution for object localisation with a mobile robot based on RFID technology
3. Vision-based approach for estimating the level of activity of a person;
4. Entropy-based system for detecting anomalous motion patterns at home;

1.3 Related work

ENRICHME In the following lines I will try to present the work related with the ENRICHME program focusing first on the state of the art on AAL, and second on the Radio Frequency Identification. Ambient Assisted Living aims to provide new technologies for helping people to remain independent, based on ambient intelligence. The goal is to create infrastructures for life scenarios. For instance, the ALL architecture introduces the CASAS “smart home

in a box" which is a smart home design that is easy to install and provides smart home capabilities out of the box with no customisation or training. [8] This allows smart home technologies to achieve its benefits at a large scale. CASAS has three main conceptual levels. First, the physical layer and the lowest level contains hardware components including sensors and actuators and its architecture utilizes a Zigbee wireless mesh which communicates directly with the hardware components. Second, the middleware layer is governed by a publish/subscribe manager. Moreover, this paradigm acts as a memory system, the manager providing named broadcast channels that allow component bridges to publish and receive messages. In the end, an application layer lies on the top which allows control to move down from the application layer to the physical components that automate the action. In Europe, one of the most important framework is UniversAAL an open platform intended to facilitate the development, distribution, and deployment of technological solutions for ALL. It consists of an extensive set of resources (some are software and some are models/architectures) aimed to benefit end users (i.e., assisted persons, their families, and caregivers), authorities with responsibility for AAL, and organizations involved in the development and deployment of AAL services. The resources are classified into three main groups: runtime support, development support, and community support. UniversAAL offers a semantic and distributed software platform designed to ease the development of integrated AAL applications. It makes use of a complementary middle complementary middleware, called open- HAB3, to control various domestic sensors. [10] Radio Frequency Identification (RFID) are a common and useful tool in recent AAL systems. These tools are used to locate a human or an object within indoor environments. However, this is only possible by caring a small, inexpensive tag. RFID localization is an interesting technology that uses classical techniques such as Time of Arrival, Time Difference of Arrival, Received Signal Phase or Trilateration, based on radio-frequency propagation models, including physical aspects of RFID communication. [6] There are various approaches for locating an object in an environment. Huiting and other professionals propose statistical models of a tag detection event itself. [12] Moreover, Soltani and Motamed propose proximity-based methods of creating, sharing, exchanging and managing the building information throughout the lifecycle among all stakeholders. [22]

Even though a lot of methods have been proposed, RFID technology is more helpful when it is necessary to monitor an activity. Most of the research have concluded that accelerometers have been widely considered as practical sensors for wearable devices to measure and assess physical activity of people. Ravi and others concluded that activities can be recognized with fairly high accuracy using a single triaxial accelerometer. Activities that are limited to the movement of just hands or mouth (e.g brushing) are comparatively harder to recognize using a single accelerometer worn near the pelvic region. [20] Atallah uses a light- weight earworn accelerometer to categorize the daily activities of people into four levels: very low (e.g., sitting), low (e.g., reading), medium (e.g., preparing food), and iv) high (e.g., sports). However, using video-based approaches are less intrusive and efficient solutions to analyse human activity. The market offers also some benefits for video based approaches since we have observed a decrease in the prices for video cameras and there is a growing interest of video-based methods to provide a smart environment for the elderly. Pal and Abhayaratne propose a framework with two different variants of the motion features captured from two camera angles and classifies them into different activity levels using neural networks. [18] Optical flow vectors are used to detect regions on the image that represents a motion. Furthermore histograms of oriented gradients features are extracted in these regions and finally, a neural network is trained to classify the level of activity. However, this came with one inconvenient the noise of the optical flow, and thus, extracting histograms of oriented gradients features can be expensive. The ENRICHME project has a different approach detecting the activity level of the person by analysing the global and local motion of the whole body and of some body parts. It is also very important to assess the overall motion behaviour of the person over extended period of times. In some cases, the detection of anomalous activity levels could indicate different health conditions. Different states of agitation could show dementia or cognitive impairments. For example, Elderly individuals

and dementia patients commonly experience disrupted sleep-wake cycles, which may lead to psychomotor agitation, confusion, and wandering. Sundowning syndrome, which encompasses many of these behaviours, is characterized by a temporal pattern in the severity of symptoms, usually expressed as worse during the late afternoon or evening. This could easily be observed in motor activity level and circadian rhythms. [3] However, the research on this subject is quite limited to research on animals and there are few examples of this syndrome monitoring on real patients. A couple of solutions based on motion trackers were proposed. Aran and others have adopted a cross-entropy metrics method to evaluate the capability of a model of predicting behavioural changes, defining an abstract layer to create a common ground for different sensor configurations. [2] This approach has inspired the method used by ENRICHME project.

Chapter 2

Basic Concepts

In the previous chapter I presented a short introduction of the ENRICHME project and subjects of work. Before I present a more detailed description of the main goals and the implementation of them I would like to discuss about the basic conecepts of working with robots and of the tehnologies and programing languages used.

2.1 Thiago and sensors

Tiago robot was developed by Pal Robotics. It has several sensors that allows it to navigate in the environment and interact with the humans. The touch monitor has been integrated on the frontal part of the robot with a support provided of a rotational joint that allows the user to adjust the screen orientation. The RFID antenna to locate objects labelled with RFID tags has been also integrated on the frontal part of the robot, just below the touch monitor. The robot has also a lifting torso. In this way the touch monitor height can be adapted to the location of the end users depending on their height and whether they are standing or sitting. Finally, the thermal camera has been integrated on top of the head to be as close as possible to the onboard RGB-D camera.

2.2 ROS

Since the scale and the goal of the robotics have constantly expend in the last few years, creating software for robots had become more complicated for software architectures. A large amount of research was made creating new software systems in response to the old weaknesses of other softwares. Moreover, new softwares were created to improve aspects that seem to be more significant than others in the design process. ROS represents a new framework for all type of robotics software. It does not act as a traditional operating system for management and scheduling. Instead, it uses a structured communication layer above the host operating systems of a heterogenous compute cluster. Initially designed to solve some specific issues in developing large-scale service robots, ROS generated a more comprehensive architecture which is far more universal than the service-robot and mobile-manipulation fields. In the following line I will present the design goals of ROS trying to describe them in a detailed manner.

1. Peer-to-peer - Using a system constructed by using ROS represents a number of processes linked at runtime in a peer-to-peer topology. Since running a central server usually results in unneeded traffic flowing across the wireless link, peer-to-peer together with "fanout"

software modules, keep away completely this issues. This topology necessitates a lookup mechanism (name service/ master) for allowing process to spot each other when running.

2. Multi-lingual - Since many professionals have diverse linguistic preferences when programming which results in various compromises, ROS was designed with a neural structure and it sustains 4 languages: C++, Python, Octave, and LISP. Moreover, ROS deals with negotiation and configuration in XML-RPC and the implementation can be found in the most common languages. For a better using of each language, ROS is natively implemented in each target language. Furthermore, to strengthen cross-language development, the operating system uses IDL (language neutral interface definition language) illustrate the messages sent between modules. This simple language-neutral interface permits the formation of messages and uses short text files to describer domains of each message. While messages are sent and received, ROS automatically manages to serialise and deserialised native implementations using code generators. Since the messages are automatically generated, it is easier list new sort of messages and this represents a "time-saver" method for the programmer.

3. Tools-based - Instead of building a monolithic development and runtime environment, the designers of ROS have chosen microkernel design with a relatively great number of small tools that build and tun various ROS components. Moreover, the components manage to fulfil different tasks, such as: visualize the peer-to-peer connection topology, measure bandwidth utilization, graphically plot message data etc.

4. Thin - Since almost all the robotics software projects incorporate drivers or algorithms easy to be used again, these cods became intertwined with the middleware. This makes more hard to release its functionality and re-use the code in a context independent of its original conditions. In contrast, ROS proposes an algorithm development which takes places having no dependence on the operating system. This "thin" ideology is possible by constructing modular builds inside the source of code three. ROS places virtually all complexity in libraries and it creates small executables which expose library functionality to ROS. This allows an easier code extraction and a reuse beyond its original intent. Moreover, ROS re-operates code from various open-source projects, like drivers, navigation system, and simulators from the Player project. In each case, the operating system can be used to show different configuration options or to route data into and out. Furthermore, the operating system is self-executing by updating source code from external repositories, apply patches, and so on.

5. Free and Open-Source - The designers of ROS made it publicly available for creating a fully open platform. The operating system can be used for not only for non-commercial projects, but also for commercial ones. It does not require for modules to be connected together in the same executable and by using inter-process communications it allows the data to be transferred between the modules.

Presenting the design goals of the new operating system, my hope is that I offered a clear perspective about how ROS has improved the perspective over the robotics software. Moreover, I will try to highlight the fundamental concepts regarding the implementation of the system which are: nodes, messages, topics, and services.

Nodes represent processes which perform computation. Being designed as a modular system with a fine-grained scale, for ROS the term "node" could also be replaced with the term "software module". Those nodes exchange information through **messages** which represent a strictly typed data structure. Standard primitive types (integer, floating point, boolean, etc.) are supported, as are arrays of primitive types and constants. Messages can be composed of other messages, and arrays of other messages, nested arbitrarily deep. To send a message, a node publish it to a given **topic**, which is simply a string like a map. The appropriate topic is automatically assigned to the specific node. However, since **publishers** and **subscribers** do not recognise

each other, they could coincide to a single topic and, a node could publish or subscribe to numerous topics. Moreover, the "broadcast" routing scheme of publish-subscribe model does not represent a suitable method for synchronous transactions which can simplify the design of some nodes. This in ROS, is called a **service** which is defined by a string name and a pair of strictly typed messages.

For more detalis a good paper to read is find at [19].

2.3 WebRTC

WebRTC (Web Real-Time Communication) [4] allows browsers share and exchange information in real time operating by a peer-to-peer communication. This gave to web developers the opportunity to assemble real-time multimedia application without proprietary plug-ins. It extends the web browsing model. To set up and manage good and well grounded communication channel between web browsers, the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) are together defining the Java Script APIs (Application Programming Interfaces), the standard HTML5 tags, and the underlying communication protocols. Running on any device through secure access to the input peripherals, is the standardisation goal of WebRTC.

Web Architecture The web architecture is founded on a client-server paradigm. The browsers send an HTTP request for content to the web server and the server answer with the information that was demanded. The resources given by a server are correlated with an entity known by a URI or URL. Moreover, the server can set the JavaScript code in the HTML page it send back to the client. Users can interact with the code by user interface and browsers can interact with the code by using standard JavaScript APIs.

WebRTC introduces the peer-to-peer communication system between browsers and in this way it expands the client-server semantics. The most common WebRTC model extract its ingenuity from SIP (Session Initiation Protocol) Trapezoid (RFC3261) where both browsers are running a web application downloaded from a different web server. To set up and put a stop to the exchange of information there are used singling messages which are transpired by HTTP or WebSocket protocol through web servers that can modify, translate, or manage them. Being considered a part of the application, the signalling between browser and server is not standardised in WebRTC. Moreover, a peer connection permits to the data to move along the browsers independent of servers. The two web servers can communicate using a standard signaling protocol such as SIP or Jingle. In WebRTC, the trapezoid transforms into a Triangle when both browsers are running the same web application.

WebRTC in the Browser Through the standardised WebRTC, the web application interacts with web browsers, allowing it to utilise and control the real-time browser function. The WebRTC web application also interacts with the browser, using WebRTC and other standardised APIs, both proactively and reactively. Thus, the WebRTC API has to have a wide set of functions, such as encoding/decoding capabilities negotiation, media control, selection and control etc. What is the most important fact concerning WebRTC is that it creates a real-time flow of data which allows direct communication between two browsers without any other intermediaries along the path.

Signaling The basic idea that lies behind the design of WebRTC is that it specify how to control the media plane, while leaving the signalling plane as much as possible to the application

layer. The reasons behind this is the fact that different application might have a preference in using different standardised singnaling protocols.

The most important information that necessitates to be exchanged is represented by session description and it identifies the transport and the media type, along with all the media configuration parameters necessary to create a media path. The IETF is standardising the JavaScript Session Establishment Protocol JSEP which provides the necessary interface to manage the negotiated local and remote session descriptions. Moreover, in this context the application receives all the responsibility for driving the singling state machine. In fact, instead of simply forwarding to the remote side the messages emitted from the browser, the application must call the right APIs at the right times to convert the sessions descriptions.

WebRTC API The W3c WebRTC API allows a JavaScript application to exploit the novel browser's real time abilities. Moreover, all media and data streams are encrypted and using real-time browser function allows the application to establish the necessary audio, video, and data channels. To prevent eavedropping, tampering, or message forgery to the datagram transport offered by User Datagram Protocol (UDP), the Datagram Transport Layer Security (DTLS) was designed which is based on the stream oriented Transport Later Security (TLS) protocol and provides similar security guarantees. The DTLS connection between two WebRTC clients is made by self-signed certificates and these certificates cannot be used to authenticate the peer. Furthermore, the IETF works on selecting a minimum set of mandatory to support audio and video codecs for ensuring a baseline level of interoperability between different real-time browser function implementations. The mandatory are Opus (RFC6716) and G.711 and they implement audio codecs, not video codecs. The API was designed to be based on three main concepts: MediaStream, PeerConnection, and DataChannel.

MediaStream A MediaStream is an abstract representation of an actual stream of data of audio and/or video and it manages actions such as displaying content, recording it, sent it to a remote peer. A LocalMediaStream is a media stream from a local media-capture device. The application must request access from the user to create and use a local stream specifying the type of data. Moreover, the devices selector in the browser interface serves as the mechanism for drafting or denying access. If the application is done, it could revoke its own access by calling the stop( ) function on the LocalMediaStream. Media-plane signaling is carried out of band between the peers. The Secure Real-time Transport Protocol (SRTP) and the RTP Control Protocol (RTCP) is used to transport media data. DTLS is used for SRTP key and association management.

PeerConnection This type of connection permits to users to share and exchange information directly, browser to browser. It represents an interconnection with the remote peer, which is usually another instance of the same JavaScript application running at the remote end. A scripting code helps at coordinating communications. If a peer connection is established, then media stream can be sent directly to the remote browser. In addition, the Session Traversal Utilities for NAT (STUN) protocol (RFC5389) allows a host application to discover the present of a network address translator on the network. Furthermore, it obtains the allocated public IP and port tuple for the current connection, but it must have assistance from a configured, STUN server which resides on the public network. The Traversal Using Relays around NAT (TURN) protocol (RFC5766) allows a host behind a NAT to obtain a public IP address and port a relay server residing on the public internet. Moreover, the host can receive media from any peer. As I presented, the PeerConnection mechanism uses STUN and TURN servers, together with ICE protocol, allowing UDP- based media streams traverse NAT boxes and firewalls. ICE acts like a security guarantee, and in the same time it allows the browsers to discover enough information about the topology of the network. The signalling messages are

fed into the receiving PeerConnection upon arrival. The APIs send signalling messages that most application will treat as opaque blobs. However, these signalling messages have to be moved in a secure and efficient manner to the other peer.

Data Channel The DataChannel API is designed for browsers in order to allow them to exchange generic data in a bidirectional peer-to-peer fashion. The standardisation work within the IETF has reached a general consensus on the usage of the Stream Control Transmission Protocol (SCTP) enclosed in DTLS to handle non media data types. The circuit presented above provides a NAT traversal solution, confidentiality, and source authentication. Furthermore, using this method the transport of data works smoothly with the parallel media transports. SCTP has been chosen because it supports numerous streams and it provides the possibility of opening several independent streams towards a peering SCTP endpoint. Each stream actually represents a unidirectional logical channel providing the notion of in-sequence delivery. A message sequence can be sent either ordered or unordered and the message delivery order is maintained only for all ordered messages sent on the same stream. Even though, the DataChannel API is designed as a bundle of an incoming and an outgoing SCTP stream. The DataChannel setup is carried out when the CrateDataChannel( ) function is called for the first time on an instantiated PeerConnection object. Each subsequent call to the CreateDataChannel () function just creates a new DataChannel within the existing SCTP association.

More details about the WebRTC [13], STUN [14],TURN [15], ICE [21].

2.4 RSS 2.0 Feed

RSS 2.0 (Really Simple Syndication) [5] is know as a tpe of web feed [9]. A web feed (news feed) is a data format to provide information which frequently change their contents. It create a plublisher-subscribers system.The publisher create a web feed and the users subscribe to it. Mostly the rss feed is used for news media to send micro-news to their customers. with this tehnology you can create a news agregator where you have multiple rss feeds from different sites and you filter them for your users. They can see a short description to the news, and after that they can have the option to go to the entire news on the parent site.
There are two types of web feed most used: rss feed and atom feed. I used rss feed because is implemented by most of the news sites and I wanted a big base of news. Because of the browsers same origin policy (Section **??**) I had to use an API that transform rss feed in jsonp.

Same Origin Policy Same Origin Policy [16] is a concept used in web application security model. In browsers which respect this policy you are not allowed to use data in scripts from pages that do not have the same origin. This policy must be respect so you will not have malicious code running in your browser or your webpage. Origin is defined as concatanation of the protocol used, hostname and port of a website. You can bypass this restriction by using the cross-origin communication with JSONP [17].

2.5 JavaScript

"JavaScript is an interpreted programing language with object-oriented (OO) capabilities. Syntactically, the core JavaScript language resembles C, C++, and Java, with programing constructs such as the if statement, the while loop, and the && operator. The similary end with this syntactic resemblance, however. Javascript is a loosely typed language, which means that variables do not need to have a type specified. Objects in JavaScript map property names to arbitrary property values. In this way, they are more like hash tables or associative arrays(in

Perl) than they are like structs (in C) or objects (in C++ or Java). The OO inheritance mechanism og JavaScript is prototype-based like of the little-known language Self. This is quite different from inheritance in C++ abd Java. Like Perl, JavaScript is an interpreted language, and it draws inspiration from Perl in a number of areas, such as its regular-expression and array-handling features" [11]
I used JavaScript in my project as the frontend language , but also as backend of the authentication and signal sever. JavaScript is an easy to use programing language and it has more features and developing than php. Another feature is that is more client-side than php. In the next subsections I will present some framework used in my developing.

JQuery JQuery is a JavaScript library which make thinks like HTML (Hyper Text Makeup Language) document traversal and manipulation, event handling, animation, and Ajax more easy-to-use. It is compatible with most of browsers and also it has behind it a big comunity. I used Jquery because it make the developing of websites fast and easy to understand by outher developers, but some parts of the code was done in pure JavaScript.

SOCKET.IO SOCKET.IO is a JavaScript library (frontend and backend APIs) used for making realiable real-time web application. You used it for making a connexion beetween the webpage given by the webserver to the backend server, mostly for message from backend server. It can use two methods. The first one used websockets that makes things fast and friendly with network bandwidth. Second it used pooling that is the clasic way to ask the server from time to time if it has something to send to you.

Sweet Alert 2 Sweet Alert 2 is JavaScript library and also a css file, which make the alerts from the browser customizable. We need this library because the application is mostly used by old people we need to have alerts that have bigger text and are more center in the webpage. Also for the call part you cand have multiple option for responding the alert.

Node.js Node.js is open-source JavaScript run-time environment that allow to write server-side JavaScript code and it is saw as on of the fundamental elements of "JavaScript everywere" paradigm. This make the client-side code and server-side code to be write in the same programing language that makes it more easy-to-understand for both parts. I used it for the server-side of my web application. I used also some extra library with it like SOSCKET.IO, JWTOKEN and some paradimgs like REST. Node.js is a fast as performance and also as time of developing, and because is JavaScript there is easy for other developers to use.

Roslibjs Roslibjs is the standard JavaScript library for interacting with ROS systems from a browser. It connects with rosbridge trough WebSockets and has an API that cand let you do things like publishing, subscribing, calling services and most of the essential ROS functionality. Because a big part of my project is the web application in a browser which run on a system with ROS I had to use as my only way to comunicate with ROS and other parts of the big project

2.6 Python

" Python is a general-purpose, high-level programming language whose design philosophy emphasizes code readability. Python's syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C, and the language provides constructs intended to enable clear programs on both a small and large scale. Python supports

multiple programming paradigms, including object-oriented, imperative and functional programming styles. It features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl and Tclm and has a large and comprehensive standard library. Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

CPython, the reference implementation of Python, is free and open source software and has a community-based development model, as do nearly all of its alternative implementations. CPython is managed by the non-profit Python Software Foundation. " [23]

Rospy Rospy is a Python client library that offer an API for interacting with ROS systems. It allows you to do interface with topics, services and parameters. The design of rospy is very easy for programers to use and take a less time in developing. This makes rospy a very good aproach for prototyping in ROS. Some of the ROS tools are made with rospy.

2.7 MongoDB

MongoDB [7] is a document-oriented and general-purpose database. I used it because its documents are in fact BSONs (similar to JSONs) which are very easy to use in JavaScript and Python. Also MongoDB is know to be a scalable database, so it makes easy implementation of the chat part, where scalability is a must. On the server-side I used it for keeping the user credentials, information, chat messages and connections list, on the robot-side for keeping the news preferences in newspapers and news categories of the user. Because I use Node.js, but also Python, I had to use some libraries, that I will present next.

PyMongo PyMongo is a Python distribution that make a connexion between Pyhton and MongoDB. It is easy to use by developers and easy to understand. Also you can transform data recived from the database in dictionaries in Pyhton.

MongoDB Node.js Driver For Node.js we have an official MongoDB driver that provides both callback and promise interaction. This driver is easy-to-use and easy-to-understand by all MongoDB users, because it use almost the the same interface.

Chapter 3

Architectural Overview

With a good understanding of basic concepts of the tehnology used we can go foward and talk about the architectural of the entire system. I will talk separetly the two apps.

Before going into presentation of every module architecture, the ENRICHME project architecture must be understand.

3.1 Social Module Architecture

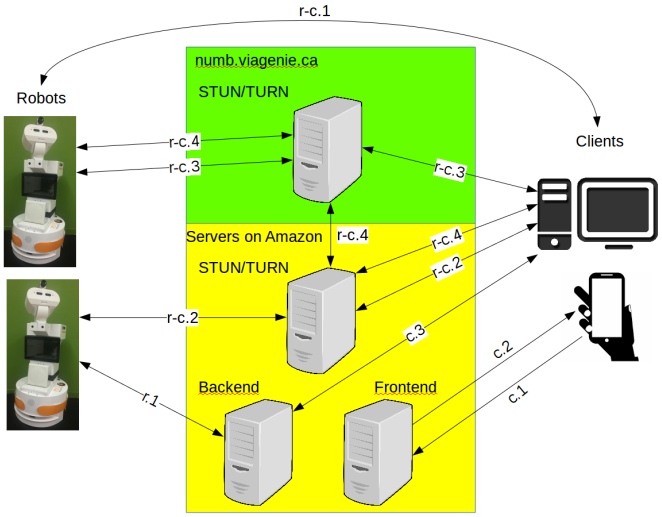

Figure 3.1: Social Module Architecture

The Social Module main objectif is the social interaction between the robot users and their relatives seen as clients users. Second objectifs are: interactions robot-robot or client-client, working under restrictive networks, performance of the video-conference call. A visual representation of the system parts and comunications is Figure 3.1.

Let's start talking about the entire workflow of calling, but firstly we will use some notation: Alice for the client or robot that is starting the call and Bob for the client or robot reciving the call. If Alice wants to call Bob she cand do that only if he is online. If Bob is online than Alice send a message different from normal messages telling Bob that she wants to start a

call containing also a RTC offer (includes some basic networking information about Alice and maybe some ICE candidates) in the same she set her local description with this offer description and she starts to gather ICE candidates and send them to Bob when she finds them. Now Bob can refuse the call and send back to Alice a reject message and then both stops for having a call or he can accept the call store de and send back to Alice a message telling that he accept the call, set his remote description with the RTC offer description just received, and creates an RTC answer that has almost the same information as the offer, set this answer as his local description and send it to Alice, in the same time he starts to gather ICE candidates and send them to Alice. Alice recive the RTC answer and set it as remote description.

After some ICE candidates changes between Alice and Bob if they do not find any matches for a RTC connection they will closed it. If they found they will go to connection established state and they will start to transmit audio/video data. They are three types of connection posible. First one if Alice and Bob are in the same network or both have public ips they cand make a direct connection (r-c.1). Second one if Alice or Bob are under a different NAT network they can make a direct connection with the help of a STUN Server or they comunicate through a common TURN server (r-c.2 adn r-c.3). Third one if Alice and Bob are under a different NAT network and they do not have a common TURN server accesible but the servers are accesible between them they cand make a connection trought two servers (r-c.4). At one moment if Alice or Bob wants to end the call they send a stop message, After both close the connection.

The SM is divided in the next parts:

- Backend server
  - REST API - offer services like creating a new user, adding new contacts to the contacts list of an user, basics information about a user, authentification, tokens, old messages between users, contacts list
  - SOCKET.IO - offer service like sending normal messages or special messages needed to make RTC connection between users and keeping an online/offline track of the users
- MongoDB database - store the collections that I use for the project . In SM architecture the MongoDB database is also on the backend server, but if the system can be implemented with the database on another server or on multiple servers agregate for more stability and failure response.
- STUN/TURN servers - used for making connections between users that transmit video/audion data.
- Frontend - the web application.

Backend Server The backend server is a Node.js server which runs on two ports: 80 for robot-side and 443 for client side with ssl certificate. It use MongoDB Node.js driver for connecting to MongoDB database. The commands are similar with the one MongoDB shell. For the REST API part there are two types of request that contains a token (created with jsonwebtoken "jwtoken", that contains username, ObjectId) or not. These are the features you can use:

1. Creating users - a HTTP/HTTPS POST message (no token) with all information about the user (username, first name, last name, gender, user type, password). First it creates the entry in credentials collections with the username and hash password after it use bcrypt-nodejs hash function on the password, after succesful insert (if the username does not exist in database) it get an ObjectId. It use the ObjectId with the outer datas from the message and create an entry in users. After it return a succesful response.

2. Authentification - a HTTP/HTTPS POST message (no token) with username and password. It use bcrypt-nodejs hash function on the password and verify it is the same hashpassword in credentials. If is not the same it return a error message, else it return the token.
3. Get information about users - A HTTP/HTTPS POST message (with token) with username or user ObjectId. The server verify if the token is valid. If is valid it take from it information about the user which made the request. After that it can easy look up if the usr from the token is connected with the user in the request body. If they are connected then it return the information aboit him.
4. Add a new connection - A HTTP/HTTPS POST message (with token) with username. The server verify if the token is valid. If is valid it take from it information about the user which made the request. Afer it use this information to connect the two user in the database (adding the usernames in connections_list array in connections_list collection)
5. Get the connections list - A HTTP/HTTPS GET message (with token). The server verify if the token is valid. If is valid it take from it information about the user which made the request. It use tis information for getting his connections_list array in connections_list collection.

For SOCKET.IO part there are two types of messages normal between users for the chat part and special messages for signaling in WebRTC. Both messages use token for authentification. The namespaces are:

- /online - After a user use the REST APi for authentification and he receive the token it connect to the server and send his token on the online namespaces to anounce that he is online. The server anounce all his connections that are also online that the user has gone online on the namespace **connection/online**. Also the server store in hashmap the socket id with the username as key and in another hashmap the username with socket id as key. When a user disconnect from the socket the entries in the hashmap are deleted. Also their connection which are online are anounce trough namespace **connection/offline** that the user has gone offline.
- /getConnections - on this namespace the user send his token and it receive back the connections that are online int that moment.
- /message/post - on this namespace the user send his token, the message text and the destinatation username. The server store the message and after that if the destiantion user is online it send the message on namespace **message/receive**.
- /message/get - on this namespace the user send his token, remote user, skip number, and limit number. It receive back the last messages order by time between him and remote user skining the skip number of messages in the limit of limit number.
- videoCall/startCall - on this namespace the user send his token, remote user and RTC offer. If the remote user is online the server send the remote user the message on the same namespace.
- videoCall/startForceCall - similar to startCall, but used in special condition that I will talk later in this paper
- videoCall/acceptCall - the same as startCall, but instead of RTc offer he has a RTC answer
- videoCall/stopCall - the same as startCall, but it has not a RTC offer
- videoCall/iceCandidate - the same as startCall, but instead of RTc offer he has a RTC ice candidate

- videoCall/restartIceOffer - the same as startCall, used for restaring ice gathering
- videoCall/restartIceAnswer - the same as acceptCall, used for restaring ice gathering
- videoCall/rejectCall - the same as stopCall
- videoCall/forceRejectCall - the same as stopCall

Because of the online/offline feature implemented in SOCKET.IO this system is limited at only one backend sever. Changing this feature in MongoDB database but making more a pooling from the clients and robots will incrase the scability of the backend server, but also the will incrase the bandwidth used and decrease performance when the system has less number of users than 40000. Because this number will not be reach in the project ENRICHME I used this type of architectural that makes the implementation more easy to understand and developing. The backend server run on two ports 80 and 443.

STUN/TURN servers STUN/TURN servers are used for making connections between users that transmit video/audion data. Because of some policy of firewalls or proxy is best to have more of this types of servers. I used on made by me on the amazon coud system and a public one, but free, server from numb.viagenie.ca. The role of STUN is that the clients can discover their network information about them if they are under a NAT network. The TURN servers come in used when users can not make a direct connection and they react as a relay server for their data. The config file for my STUN/TURN server on Amazon Cloud is here.

Webpage application The Webpage application for the Social Module is made with HTML/CSS and JavaScript. The SM design is modular with a central object which makes the connections betweens JavaScript modules. The name of the object is lic_comunicator. For integrating the webRTC and reuse of the code the thesis propose a new framework that provide a more simple interface for developers. The framework is represented by a single object called lic_call_object. This object use webRTC in background and has the next functions:

- constructor - Takes two parameters: the remote user username and the lic_comunicator object
- getLocalVideoPromise - Create and return the local video stream.
- closeLocalVideo - Close the local video stream.
- getRemoteVideoPromise - Return the remote video stream
- call - Call the remote user
- forceCall - the same as call, but this method force the remote user to answer.
- accept - Accept the incoming call.
- hangup - Close the video-conference
- reciveHangup - the same as hangup.
- resetCall - reset the object.
- addIceCandidate - Add an ICE candidate gived as parameter to function to ice candidates pool.
- setRemoteDescription - The description gived as parameter to function is seted as remote description.
- reciveRemoteDescription -The description gived as parameter to function is stored inside object, but is not seted as remote description.

- acceptIceRestart - accept and start an ICE gathering.
- reciveRejectCall - Close the video-conference if the call is rejected by the remote user
- rejectCall - Close the video-conference and send a reject message to remote user
- destructor - destroy and free the object.

This module can be changed with another JavaScript module that provide the same functions which can use a different tehnology than webRTC without affecting the rest of the app. Another JavaScript module proposed by this thesis is the lic_socket object, Its role is to comunicate with the backend server. It tuse the SOCKET.IO, but the tehnology can be changed without affecting the rest of the app. The object provide the next functions:

- constructor - Takes two parameters: the backend server url and the lic_comunicator object and creates the object and starts a connection with the backend server.
- emit - send the a JavaScript Object (given as the second parameter) to a namespace on the backend server (first parameter) and call a function (third parameter) on the answer recived from the server.
- close - close the connection with the backend server
- Some event handler for messages recived from the server.
  - recived message for the current user from another user
  - message about a user going online
  - message about a user going offline
  - a start call offer from another user
  - a force start call offer from another user
  - an accept for a start call offer
  - a stop call message
  - an remote ice candidate
  - a restart ice gathering offer
  - an accept for a restart ice gathering offer
  - a reject for a start call offer
  - a force reject for a start call offer

The lic_comunicator works only with messages sended to him with the help of his function addMessage. It was some variables:

- socket - lic_socket object type;
- callObjects - an hashmap with lic_call_object where the key is the username of the remote user
- currentState - the state of SM("chat", "video-conference", "login", "signup", "home", "outside"(outside of the SM))
- currentRemoteUser - current remote user
- connectionsList - an array with connections
- token - the token given after authentification
- selectedUser - the current user selected

- backendServerUrl - the backend server URL

Every message sended to comunicator as type.The types are:

- "remotevideo" - return the remote video stream
- "onicecandidate" - send an ice candidate to the remote user
- "localvideo" - return the local video stream
- "callAction" - call the remote user
- "forceCallAction" - force call the remote user
- "acceptAction" - accept the incoming call
- "hangupAction" - hangup the current video-confference
- "callOffer" - sent a call offer
- "forceCallOffer" - sent a force call offer
- "acceptAnswer" - sent accept answer to a call
- "sentHangup" - send a hangup message to remote user
- "restartIceOffer" - send a restart ICE candidate gathering offer
- "restartIceAnswer" - send a restart ICE candidate gathering answer
- "reciveCall" - react to receiving a call offer
- "reciveForceCall" - react to receiving a force call offer
- "reciveAccept" - react to receiving a accept answer
- "reciveStopCall" - react to receiving a hangup message from remote user
- "reciveIceCandidate" - react to receiving a ICE candidate from remote user
- "reciveRestartIceOffer" - react to receiving a restart ICE candidate gathering offer
- "reciveRestartIceAnswer" - react to receiving a restart ICE candidate gathering answer
- "reciveRejectCall" - react to receiving a reject message from remote user
- "reciveForceRejectCall" - react to receiving a force reject message from remote user
- "destructorCallObject" - destroy the lic_call_object of a user given
- "constructorCallObject" - create the lic_call_object of a user given
- "changePage" - change the page
- "sentRejectCall" - send a reject message to remote user
- "setCurrentState" - set the current state of the comunicator
- "setCurrentRemoteUser" - set the current remote user
- "getConnectionsList" - get the connections list from the backend server
- "connectionOnline" - react to receiving a message about a user going online
- "connectionOffline" - react to receiving a message about a user going offline
- "selectUser" - select a user given
- "addConnection" - add a new connection
- "getRemoteUserInfo" - get informatoin about a user from backend server

- "getUserInfo" - get the current user information from backend server
- "sentMessage" - send a chat message to a remote user
- "getMessages" - get old messages from a user
- "reciveMessage" - react to receiving a chat message from a remote user
- "login" - login the user on backend server and set token with receiving token
- "signup" - signup a new user on backend server
- "setToken" - set a given token for the comunicator
- "getToken" - get the comunicator token
- "getParams" - get the params from param file of the SM
- "getTranslation" - get the translations from the lang file of the SM
- "goOnline" - send a message to the backend server for going online
- "createSocket" - create a lic_socket object
- "messageArray" - react to a sesion store array of messages for comunicator
- "sentForceRejectCall" - sent a force reject to remote user

The robots have the webpage application implemented in the hmi_bridge ROS package that is part of the big project ENRICHME, so it does need any connection to frontend server for taking the html or JavaScript scripts. It only makes a connection using SOCKET.IO to the backend server called in the Figure 3.1 r.1. This connection authentificate the robot to the server and announce the others users that is online and it can have a video call. Trough this connection the robots can recive events like when a user go online or offline, recive its contacts list, recive particulary information about some users. Some other two importants thinks that goes trough this connection are the messages and the informaton about starting a call. First the messages, the robot can send them trough r.1, if the other client or robot is online it will recive the message trough its r.1 or c.3 connection, if is offline than the message will be keep in the MongoDB database on the backend server (or in some specialised MongoDB database servers) and the recive users can request them lately trough REST API or SOCKET.IO connection. All the messages are keep in database, so the users cand see them lately. For the chat part the robots see their ROS parameter for language and makes the right keyboard for the user. Secondly the video-confference part, the robots use their r.1 for transmmiting a call request or to respond a reciving call, and for all the other messages related to the call app, except audio/video data. You cand send audio/video over the r.1 connection but will have a bad performance because it used TCP not UDP and also it always go trough the server. Some difference between robots and normal clients are that robots do not used the camera so the video is taken trough a ROS topic for a RGB camera and you can not add connection on the robot because of the type of users that we have on this aprt of application.

Only clients use the frontend server. The clients make a request to the frontend server for the html and JavaScript scripts (c.1) and get a them as response (c.2) and after that they make the SOCKET.IO connection with the backend server (c.3). I made the design of the page in only one html file and in more JavaScript scripts so the users will have the sensation of using an app not a webpage, and also for the performance, because after the first request the clients will not request anything more from the frontend server. This design and the fact that the project does not have a big number users the performance is high. If there it need to implement the system with less cost we can fusion the frontend server and the backend server without cost to performance too high. Because browsers let webcam and microphone share only over secure connections or locahost inscure connection (more for testing) it is a must to have a ssl certificate for the frontend servers. Because of same origin policy and security issues for browsers also the

backend server need a certificate and that is why the backend server is running on two ports 80 for the robots that are using an locahost webpage and 443 for the clients.

3.2 News Module

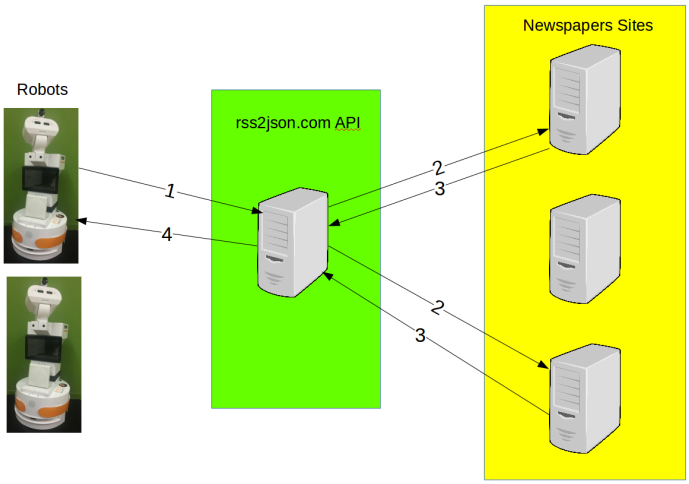

Figure 3.2: News Module Architecture

The News Module has the functionality of a news agregator. This means it takes news form different newspapers and websites with the help of a web feed (in this case RSS feed). The difference is that it use an internet API like rss2json.com which transform the RSS feed XML information into a JSONP. The JSONP can bypass the same origin policy implemented in browsers and it is easier to use within the JavaScript language. This is necesary because most of the websites today have not yet implemented RSS feed over secure connections that can be reused in outher sites. The rss2json.com API is REST API that works with GET requests that contains RSS feed url, an API key and the maximum number of news to be taken. The response of this request is a JSONP message. The information recived then it can be used in two methods:

- Writed on the webpage interface and after read by the user.
- Read by the robot.

The news can be filtred by newspaper or category. This can be done from prefferences page inside the application. The part that takes the most amount of time on this module is to find good RSS feed and find them by category. After that you complete information about them in a json file. Also it must be done for every country-language. An example of language json file is here. In this files you also have the translations of buttons or the words used in the web application.

The News Module has two pages on the web application presented on the robot:

- News Page - the user can read, listen or change the current news
- Prefferences Page - the user can change their prefferences in newspapers and categories

The architecture is modular with an central object makes the connections between JavaScript modules. The object is called lic_news_object. Togheter with the JavaScript modules and

json files creates a framework which can be used by other developers in creating their own news agregator. The lic_news_object has the next functions:

- constructor - Create the object (no parameters)
- getLang - It loads the language file given as parameter. This is necesary for translation of the page. It must be called first because some news dynamic content can use translation of words and also contains the newspapaers and categories you can access.
- getParams - It loads a params file (json format) that contains the rss2json.com API address. After it call a ROS service that connect to the robot MongoDB databse to get the last prefferences for the current user.
- getNews - It send request to rss2json.com API server for the today news that respect user prefferences.
- writeNews - Write the content of the current news on the page.
- news_prev(next) - Change the current news to the previous(next) news.
- writeToChoseNews - Write on prefferences page the newspapaers green for selected, yellow fo unselected.
- writeToChoseCategories - The same as writeToChoseNews but for categories.
- news_addNews(remove_news) - Select (unselect) a newspaper given as parameter and send the current prefferences to a ROS topic.
- news_addCategory(removeCategory) - Select (unselect) a category given as parameter and send the current prefferences to a ROS topic.
- say - The robot read the current news

For the web application to work it need an intermediary system for connecting with the local MongoDB on the robot. This means it needs a service that can be call for infomation from the database and a subscriber that saves into database data publish on a ROS topic by the web application. The ROS service and the ROS subscriber are implemented in python and starts running in the same time as the web application.

3.3 Heart Rate Bluetooth Sensor Module

Heart Rate Bluetooth Sensor Module (HRBSM) objectif is to publish in real-time data from the sensor into a ROS topic. This means that it is a ROS publisher. For getting data from the user it need to have access to a Bluetooth adapter. For this it use the gatttool, a UNIX tool. It scans for BLE devices, after a timeout it search into the list of devices and finds the heart rate sensor. It connects to it. It finds its characteristics and search for the address where the device is publishing its heart rate data. It subscribe to that address and send a message to the device to start giving information about the heart rate. On the address subscribed it creates a callback that takes the data and publish on a ROS topic togheter with the current UNIX time stamp. This is the normal workflow. The most amount of time the Module handle the problems that can raise trough the connection. For example:

- The device disconect. The HRBSM try to recconect first if the problem persist it restart the adapter and try to recconect again. If this also does not work it try to find if there are heart rate sensors close that it can connect. If they are not than it announce on ROS topic for errors that can not find any sensors nearby.
- The device is connected, but the HRBSM does not recive any data. It disconect and after it try to recconect.

- The adapter go down. The HRBSM try to restart it. If it does not work it announce on a ROS topic.
- The connection to the ROS topic break. The HRBSM try to remake the connection. If it does not work it stops and write on the log file the error.

Thesis provide a class which has the next functionalities:

- constructor -
  - create a GATTTOOL adapter
  - find the BLE device Address
  - start the adapter
  - connect to the device
- start_notification -
  - find device characteristics
  - find the heart rate characteristic address in characteristics
  - subscribe a callback function given as parameter to hear rate characteristic
  - start the notification process for heart rate characteristic
- destroy - disconect device and stop the adapter
- restart - restart the connection with the device and the subscribe process

Chapter 4

User Interface

With the knwoledge of architectural we can go foward to the user interface. This is also an important part of the project because must adapt for persons with speacials needs. That is the reason why on robots you have buttons and the font-size bigger than client side of the page.

4.1 Social Module User Interface

The SM is separated in three parts. First part is the home which contains a list of you connections. On client you have also the option to add new connections. If you press any of the users names in the list it will pop-up a small window that contains information about the user and options like :

- Message - change page to chat part
- Call - change page to video-conference part
- Ok - close the pop-up

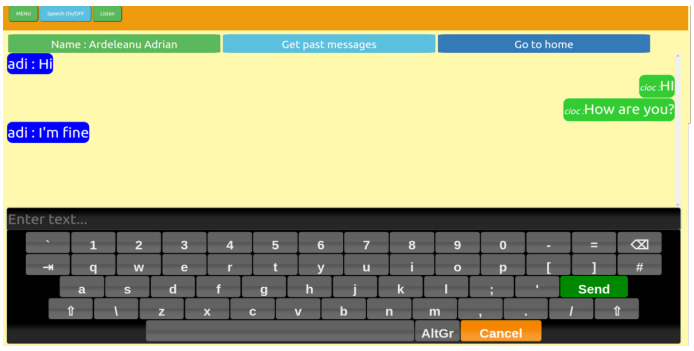

Figure 4.1: Chat UI robot

Chat The chat part on robot has a keyboard in a specific language(almost half of the screen), a box that contains the messages between users and three specific buttons:

- First button - give you more information about the remote user
- Get past messages - get messages from the past conversations

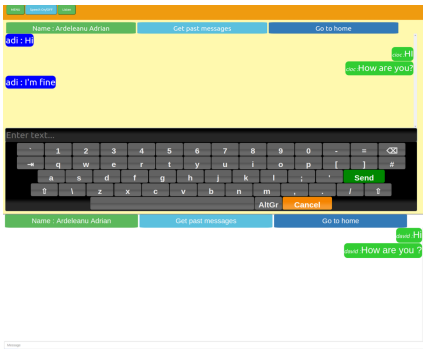

Figure 4.2: Chat UI client

- Go to home - change page to home page

The client side does not hava keyborad, therefore it has more space for messages.

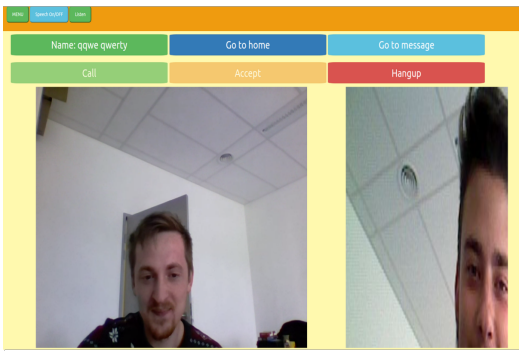

Figure 4.3: Call UI robot

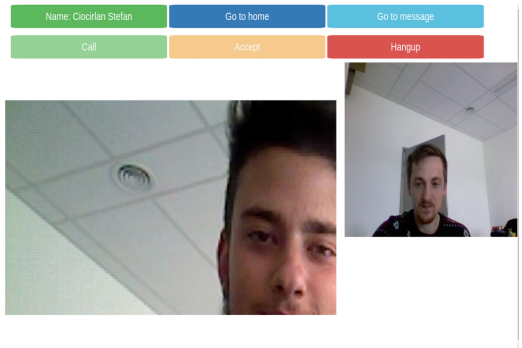

Figure 4.4: Call UI client

Video-conference The video-conference part is almost the same on robot and client (the robot has the extra menu button on top). You have on top three buttons:

- First button - provide information about the remote user
- Home - change page to home page
- Message - change page to chat part with the current remote user

Under them there are three more buttons specific to call. The three buttons are:

- Call - start a call with the remote user
- Accept - accept a receiving call
- Hangup - reject a receiving call or hang-up the current call

First, only the call button is available. If you call someone it will be diabled and the hangup button will be activated. If your call is accepted nothing will happen to avability of buttons. If your call is rejected then a pop-up will apear saying that you call was rejected by the remote user and the avability of buttons it will be the one on the start. If you are reciving a call then the call button will go disabled and will have availabled the accept and hangup button. Under this buttons are your local video (your image from the camera, on the right) and remote video (remote user image from their camera, on the left)

4.2 News Module

Figure 4.5: News UI news

Figure 4.6: News UI prefferences

The NM has two parts: the first part where the news are showed to the user and and the second one where the user can change his prefferences about newspapars and categories of news. The News Module is presented only on robot side.

News The news part has four buttons on top and the news content on the rest of the page. The buttons are:

- Previous button - change the current news content with the previous one content

- Next button - change the current news content with the next one content
- Read news button - makes the robot read your news content description
- Prefferences button - change your page to prefferences where you can chose newspapers and categories

Prefferences The prefferences page has:

- A list with newspapers that you can select because they are buttons. They have two colors green for selected and yellow for unselected
- A list with categories that you can select because they are buttons. They have two colors green for selected and yellow for unselected
- Back button - change your page with news page

Chapter 5

Implementation

The thesis present next the implementation of the modules.

5.1 Social Module Implementation

Implementation of the next parts will be explained:

- MongoDB database
- Backend server
  - REST API
  - SOCKET.IO
- Frontend

5.1.1 MongoDB database

I use a MongoDB version 3.2.10 database. I created a new database call LIC and inside it I use the next four collections:

- credentials - This collection contains the username, the hash password, the ObjectID for the user and the time when the acount was created. I used it for authentification.
- users - This collection contains basic information about the users like :
  - ObjectId
  - Username
  - Gender
  - Last name
  - First Name
  - User Type - (Doctor, Client, Family)
- connections_list - This collection contains the contacts of the users. The document has the next informations:
  - ObjectId

  - Username
  - connections_list - An array of usernames of the contacts
- messages - This collection contains the messages between users. The document contains the next informations:
  - ObjectId
  - type - the type of the message (normal or special)
  - to_uid - The ObjectId of the user that received this message
  - from_uid - The ObjectId of the user that sended this message
  - to - The username of the user that received this message
  - from - The username of the user that sended this message
  - sentAt - time when it was sent
  - text - message text
  - roomId - A concatanation of ObjectIds of the users so it make easier to find messages betweens two users.

I will list the things I did for improving the performance of search/insert or making the code easy-to-understand in database:

1. connections_list array with usernames makes more easy to use because I insert usernames not ObjectIds and it is more fast for search because I get the entire array of connections with a single document and I can use directly the ObjectId provide by the token, when a request is made.
2. username in users collections make easier to understand the code when I search after the username instead of ObjectId
3. roomId in messages boost a lot the performance because I search only after a single thing in the document that is also unqiue.
4. sentAt in messages help for sorting after time the messages
5. to_uid, from_uid in messages boost performance for searching the messages of an single user
6. hashpassword in credentials - not keeping the passwords in clear text

5.1.2 Backend server

The backend server is a Node.js server. It use the next Node.js modules:

- Node.js MongoDB driver - used for connecting to MongoDB database
- fs - filesystem for opening the ssl certificate and private key
- express - for creating an express app (REST API)
- http - for creating an http server
- https - for crating an https server
- cors - for cross origin implementation
- socket.io - for SOCKET.IO implementation

The REST API implementation in pseudocod is:

1. Creating users -

```
1         if (HasNotAllInformations) then return FAIL;
2         if (UsernameExists or UsernameNotValid) then return FAIL
              ;
3         CreateCredentialsInDatabase();
4         CreateUserInformationInDatabase();
5         return SUCCESS;
```

2. Authentification -

```
1         if (HasNotUsername or HasNotPassword) then return FAIL;
2         hashpassword := getFromCredentialsHashpassword(username)
              ;
3         if (hash(password) <> hashpassword) then return FAIL;
4         userinfo := getFromUsersInformation(username)
5         token := createToken(userinfo)
6         return token;
```

3. Get information about users -

```
1         if (invalidToken) then return FAIL;
2         if (HasNotRequestUsername and hasNotRequestObjectID)
              then return FAIL;
3         if (hasRequestObjectID) then requestUsername :=
              getUsernameFromUsers(RequestObjectID);
4         else requestUsername := RequestUsername
5         if (requestUsername is not in
              getConnectionsFromConnectionsList(token_username))
              then return FAIL;
6         userinfo := getFromUsersInformation(requestUsername)
7         return userinfo;
```

4. Add a new connection -

```
1         if (invalidToken) then return FAIL;
2         if (HasNotRequestUsername) then return FAIL;
3         requestUsername := RequestUsername
4         if (requestUsername is in
              getConnectionsFromConnectionsList(token_username))
              then return FAIL;
5         addConnectionIntoConnectionsList(token_username,
              requestUsername);
6         addConnectionIntoConnectionsList(requestUsername,
              token_username);
7         return SUCCESS;
```

For SOCKET.IO implementation in pseudocod is:

- /online - user go online

```
1         if (invalidToken) then return FAIL;
2         AddInHashMap(username_socket, (token_username, socket_id
              ))
3         AddInHashMap(socket_username, (socket_id, username))
```

```
4          connections_list = getConnectionsFromConnectionsList(
               token_username)
5          for connection in connections_list do:
6              if (IsOnline(connection)) then
7                  sendConnectionOnline(connection, token_username)
8          endfor
9          return SUCCESS;
```

- disconnect - a used disconnect

```
1          username = getValueHashMap(socket_username, socket_id)
2          deleteFromHashMap(username_socket, username)
3          deleteFromHashMap(socket_username, socket_id)
4          connections_list = getConnectionsFromConnectionsList(
               username)
5          for connection in connections_list do:
6              if (IsOnline(connection)) then
7                  sendConnectionOffline(connection, username)
8          endfor
```

- /getConnections - get connections list as an hashmap where value is true if the user is online or false if he is offline

```
1          if (invalidToken) then return FAIL;
2          connections_list = getConnectionsFromConnectionsList(
               token_username)
3          responseData = new hashMap();
4          for connection in connections_list do:
5              if (IsOnline(connection)) then
6                  AddInHashMap(responseData, (connection, true))
7              else
8                  AddInHashMap(responseData, (connection, false))
9          endfor
10         return responseData;
```

- /message/post - send a message

```
1          if (invalidToken) then return FAIL;
2          connections_list = getConnectionsFromConnectionsList(
               token_username)
3          if (remote_user not in connections_list) then return
               FAIL;
4          saveToMessages(message)
5          if(IsOnline(remote_user))
6              sentreceiveMessage(remote_user, message)
7          return SUCCESS;
```

- /message/get - get old messages

```
1          if (invalidToken) then return FAIL;
2          connections_list = getConnectionsFromConnectionsList(
               token_username)
3          if (remote_user not in connections_list) then return
               FAIL;
```

```
4         messages = getFromMessages(token_username, remote_user,
              skip, limit)
5         return messages;
```

- videoCall/startCall , videoCall/startForceCall ,videoCall/acceptCall , videoCall/stopCall, videoCall/iceCandidate, videoCall/restartIceOffer, videoCall/restartIceAnswer, videoCall/rejectCall, videoCall/forceRejectCall. - special messages for WebRTC

```
1         if (invalidToken) then return FAIL;
2         connections_list = getConnectionsFromConnectionsList(
              token_username)
3         if (remote_user not in connections_list) then return
              FAIL;
4         saveToMessages(message)
5         if(IsOnline(remote_user))
6             sentSpecialMessage(remote_user, message)
7         return SUCCESS;
```

5.1.3 Frontend

The thesis presents the important objects implementation in pseudocod. The other implementation are simple and can be easy understand after a simple look over the code. First object is the lic_call_object:

- constructor - create the object

```
1         setComunicator(comunicator)
2         setRemoteUsername(remote_user)
3         setIceServers(STUN/TURN servers)
4         setRTCOfferOptions(audio and video)
5         setGetUserMediaConstraints(audio and video) //robot only
              audio
6         setHasDoneIceRestart(false)
7         setRTCPeerConnection(null);
8         setLocalStream(null)
9         setRemoteSteam(null)
10        setRemoteDescription(null)
```

- getLocalVideoPromise - Create and return the local video stream.

```
1         if(hasLocalStream) then return LocalStream;
2         stream = createLocalStream() //getUserMedia(
              UserMediaConstraints) //on robot create canvas from
              ros topic image data
3         setLocalStream(stream)
4         return stream;
```

- closeLocalVideo - Close the local video stream.

```
1         if(hasNotLocalStream) then return FAIL;
2         setLocalStream(null)
3         return SUCCESS;
```

- getRemoteVideoPromise - Return the remote video stream

```
1         while(hasNotRemoteStream) wait;
2         return RemoteStream;
```

- call - Call the remote user

```
1         if (hasRTCPeerConnection) then return FAIL;
2         RTCPeerConnection = new RTCPeerConnection(IceServers)
3         addStream(RTCPeerConnection, LocalStream)
4         RTCOffer = createRTCOffer(RTCPeerConnection,
              RTCOfferOptions)
5         setLocalDescription(RTCPeerConnection,
              RTCOffer_description)
6         SendMessageComunicator(Comunicator, 'callOffer',
              RTCOffer_description)
7         return SUCCESS;
```

- forceCall - the same as call, but instead of 'callOffer' it use 'forceCallOffer'.
- accept - Accept the incoming call.

```
1         if (hasRTCPeerConnection) then return FAIL;
2         if (hasNotreceivedRemoteDescription) then return FAIL;
3         RTCPeerConnection = new RTCPeerConnection(IceServers)
4         addStream(RTCPeerConnection, LocalStream)
5         setRemoteDescription(RTCPeerConnection,
              receivedRemoteDescription)
6         RTCAnswer = createRTCAnswer(RTCPeerConnection)
7         setLocalDescription(RTCPeerConnection,
              RTCAnswer_description)
8         SendMessageComunicator(Comunicator, 'acceptAnswer',
              RTCAnswer_description)
9         return SUCCESS;
```

- hangup - Close the video-conference

```
1         if (hasNotRTCPeerConnection or hasNotRemoteConnection)
              then return rejectCall()
2         closeRTCPeerConnection()
3         setRTCPeerConnection(null)
4         resetCall()
5         SendMessageComunicator(Comunicator, 'sentHangup')
6         return SUCCESS;
```

- receiveHangup - same as hangup

```
1         closeRTCPeerConnection()
2         setRTCPeerConnection(null)
3         resetCall()
4         return SUCCESS;
```

- resetCall - reset the object.

```
1         setHasDoneIceRestart(false)
2         setRTCPeerConnection(null);
```

```
3          setLocalStream(null)
4          setRemoteSteam(null)
5          setRemoteDescription(null)
6          setHasRemoteConnection(false)
7          return SUCCESS;
```

- addIceCandidate - Add an ICE candidate gived as parameter to function to ice candidates pool.

```
1          while (hasNotRemoteConnection) wait;
2          addIceCandidate(RTCPeerConnection,
               received_ICE_candidate)
3          return SUCCESS
```

- setRemoteDescription - The description gived as parameter to function is seted as remote description.

```
1          setRemoteDescription(RTCPeerConnection,
               received_description)
2          setHasRemoteConnection(true)
```

- receiveRemoteDescription - The description gived as parameter to function is stored inside object, but is not seted as remote description.

```
1          setreceivedRemoteDescription(received_description)
2          return SUCCESS;
```

- acceptIceRestart - accept and start an ICE gathering.

```
1          closeCurrentConnection()
2          setRemoteDescription(RTCPeerConnection,
               received_description)
3          setHasRemoteConnection(true)
4          RTCAnswer = createRTCAnswer(RTCPeerConnection)
5          setLocalDescription(RTCPeerConnection,
               RTCAnswer_description)
6          SendMessageComunicator(Comunicator, 'restartIceAnswer',
               RTCAnswer_description)
7          return SUCCESS;
```

- receiveRejectCall - Close the video-conference if the call is rejected by the remote user

```
1          receiveHangup()
2          setreceivedRemoteDescription(null)
3          return SUCCESS
```

- rejectCall Close the video-conference and send a reject message to remote user

```
1          SendMessageComunicator(Comunicator, 'sentRejectCall')
2          return SUCCESS
```

- destructor

```
1          hangup()
2          resetCall()
3          closeLocalVideo()
4          return SUCCESS;
```

- event Handler ICE connection failed

```
1          if(hasDoneICeRestart) then return null;
2          startICERestart()
```

- startICERestart - start an ICE gathering

```
1          makeRestartICEOfferOptions()
2          RTCOffer = createRTCOffer(RTCPeerConnection,
               restartIceOfferOptions)
3          setLocalDescription(RTCPeerConnection,
               RTCOffer_description)
4          SendMessageComunicator(Comunicator, 'restartIceOffer',
               RTCOffer_description)
```

Second object is lic_socket:

- constructor -

```
1          setComunicator()
2          setBackendServer()
3          connectBackendServer()
4          setEventsHandlers()
5          return socket;
```

- emit - implemented by SCOKET.IO.
- close - implemented by SCOKET.IO.
- Some event handler for messages received from the server.
  - received message for the current user from another user

```
1                  SendMessageComunicator(Comunicator, '
                       receiveMessage', received_message)
```

  - message about a user going online

```
1                  SendMessageComunicator(Comunicator, '
                       connectionOnline', username)
```

  - message about a user going offline

```
1                  SendMessageComunicator(Comunicator, '
                       connectionOffline', username)
```

  - a start call offer from another user

```
1                  SendMessageComunicator(Comunicator, '
                       receiveCall', receivedRTCOffer)
```

- a force start call offer from another user

```
1                   SendMessageComunicator(Comunicator, '
                        receiveForceCall', receivedRTCOffer)
```

- an accept for a start call offer

```
1                   SendMessageComunicator(Comunicator, '
                        receiveAccept', receivedRTCAnswer)
```

- a stop call message

```
1                   SendMessageComunicator(Comunicator, '
                        receiveStopCall', username)
```

- an remote ice candidate

```
1                   SendMessageComunicator(Comunicator, '
                        receiveIceCandidate',
                        received_iceCandidate)
```

- a restart ice gathering offer

```
1                   SendMessageComunicator(Comunicator, '
                        receiveRestartIceOffer', receivedRTCOffer
                        )
```

- an accept for a restart ice gathering offer

```
1                   SendMessageComunicator(Comunicator, '
                        receiveRestartIceAnswer',
                        receivedRTCAnswer)
```

- a reject for a start call offer

```
1                   SendMessageComunicator(Comunicator, '
                        receiveRejectCall', username)
```

- a force reject for a start call offer

```
1                   SendMessageComunicator(Comunicator, '
                        receiveForceRejectCall', username)
```

Last the lic_comunicator:

- "remotevideo" - return the remote video stream

```
1                 return getRemoteVideoPromise(callObjects(
                      username))
```

- "onicecandidate" - send an ice candidate to the remote user

```
1                 socket.emit("videoCall/iceCandidate",
                      IceCandidate, ack);
```

- "localvideo" - return the local video stream

```
1                   return getLocalVideoPromise(callObjects(username
                        ))
```

- "callAction" - call the remote user

```
1                   getLocalVideoPromise(callObjects(username))
2                   call(callObjects(username))
```

- "forceCallAction" - force call the remote user

```
1                   getLocalVideoPromise(callObjects(username))
2                   forceCall(callObjects(username))
```

- "acceptAction" - accept the incoming call

```
1                   getLocalVideoPromise(callObjects(username))
2                   accept(callObjects(username))
```

- "hangupAction" - hangup the current video-confference

```
1                   hangup(callObjects(username))
```

- "callOffer" - sent a call offer

```
1                   socket.emit("videoCall/startCall", RTCOffer, ack
                        );
```

- "forceCallOffer" - sent a force call offer

```
1                   socket.emit("videoCall/startForceCall", RTCOffer
                        , ack);
```

- "acceptAnswer" - sent accept answer to a call

```
1                   socket.emit("videoCall/acceptCall", RTCAnswer,
                        ack);
```

- "sentHangup" - send a hangup message to remote user

```
1                   socket.emit("videoCall/stopCall", username, ack)
                        ;
```

- "restartIceOffer" - send a restart ICE candidate gathering offer

```
1                   socket.emit("videoCall/restartIceOffer",
                        RTCOffer, ack);
```

- "restartIceAnswer" - send a restart ICE candidate gathering answer

```
1                   socket.emit("videoCall/restartIceAnswer",
                        RTCAnswer, ack);
```

- "receiveCall" - react to receiving a call offer

```
1                   case: currentState == callState
2                   if(InVideoConference or remote_user <>
                        from_username or reject) {
3                       SendMessageComunicator(Comunicator,
                            sentRejectCall, from_username)
4                   } else {
5                       receiveRemoteDescription(callObjects(
                            username), RTCOffer)
6                   }
7                   case: currentState <> callState
8                   if(reject) {
9                       SendMessageComunicator(Comunicator,
                            sentRejectCall, from_username)
10                  } else {
11                      makeMessageArray('changePage','
                            getTranslation','forceCallAction')
12                      store(messageArray)
13                      changePageToCall();
14                  }
```

- "receiveForceCall" - react to receiving a force call offer

```
1                   case: currentState == callState
2                   if(InVideoConference or remote_user <>
                        from_username) {
3                       SendMessageComunicator(Comunicator,
                            sentRejectCall, from_username)
4                   } else {
5                       receiveRemoteDescription(callObjects(
                            username), RTCOffer)
6                       SendMessageComunicator(Comunicator,
                            acceptAction, from_username)
7                   }
8                   case: currentState <> callState
9                   SendMessageComunicator(Comunicator,
                        sentRejectCall, from_username)
```

- "receiveAccept" - react to receiving a accept answer

```
1                   setRemoteDescription(callObjects(username),
                        RTCAnswer)
```

- "receiveStopCall" - react to receiving a hangup message from remote user

```
1                   receiveHangup(callObjects(username))
```

- "receiveIceCandidate" - react to receiving a ICE candidate from remote user

```
1                   addIceCandidate(callObjects(username),
                        receivedIceCandidate)
```

- "receiveRestartIceOffer" - react to receiving a restart ICE candidate gathering offer

```
1                   acceptIceRestart(callObjects(username), RTCOffer
                        )
```

- "receiveRestartIceAnswer" - react to receiving a restart ICE candidate gathering answer

```
1                   setRemoteDescription(callObjects(username),
                        RTCAnswer)
```

- - The one related to video-conference part are similar with ones above
- "changePage" - change the page

```
1                   makeCurrentStateDivInvisible()
2                   makeNextStateDivVisible()
```

- "getConnectionsList" - get the connections list from the backend server

```
1                   connections_list = ajax.request(username)
2                   write connections as buttons in html
```

- "selectUser" - select a user given

```
1                   userinfo = ajax.request(username)
2                   sweetalert2(userinfo)
```

The client problems were mostly because it need a secure connection so it must have ssl certificates. The robot problem was that it does not have a webcam. I used an asus camera connected to the robot. The image of the camera was published on a ROS topic. On the lic_call_object I subscribe to that topic and create canvas and I use captureStream() for creating a stream. For pop-up windows it use sweet alert 2.

5.2 News Module Implementation

For the NM the thesis presents the lic_news_object implementation:

- getLang - loads the language file

```
1           LoadJSON(lang_file)
2           setNewspapers(lang_file.news)
3           setCategories(lang_file.categories)
4           setTranslations(lang_file.translations)
5           setButtonsText(lang_file.buttons)
```

- getParams - loads the param file

```
1           LoadJSON(param_file)
2           setBackendServer(lang_file.apiServer_url)
3           callRosService(MongoNews)
4           setChosenNewspapers(MongoNews.news)
5           setChosenCategories(MongoNews.categories)
```

- getNews - It send request to rss2json.com API server for the today news that respect user prefferences.

```
1          for category in categories:
2              for newspaper in newspapers:
3                  if(category in chosenCategories and newspaper in
                       ChosenNewspapers)
4                      news = ajax.request(BackendServer,
                           newspapers[category].link)
5                      pushInArray(toWriteNews, news)
```

- news_addNews(remove_news) - Select (unselect) a newspaper given as parameter and send the current prefferences to a ROS topic.

```
1          ChosenNewspapers.push(news) //remove
2          puplishROSTopic(ChosenNewspapers + chosenCategories);
```

5.3 Heart Rate Bluetooth Sensor Module

The HRBSM use a python module called pygatt which use the UNIX tools called gatttool and hciconfig.

The module contains a class (HEART_RATE_DEVICE) for working with the device:

- constructor

```
1      SetTheBluettohInterfaceUsed()
2      SetDeviceName()
3      SetBLEAddressType()
4      createGATTTOOLAdapter(BluetoothInterface)
5      devices = adapter.scanBLEDevices()
6      find (DeviceName in devices)
7      SetDeviceAddress()
8      StartAdapter()
9      ConnectToDevice()
```

- start_notification

```
1      SetNotificationCallback(callback_function)
2      characteristics = getCharacteristicsFromDevice()
3      find (heart_rate_characteristic in characteristics)
4      subscribeNotificationCallback(
           heart_rate_characteristic_address)
5      startTheNotificationProcess() //write on bit in a register
```

- restart

```
1      If (deviceConnected and receivingDataOnsubscriber) then
           return;
2      if (deviceConnected and NotReceivingDataOnSubscriber) {
3          If(counter > MAX_COUNTER) {
4              restartAdapter()
5              count = 0;
6          }
7          disconnectDevice()
8          reconnectDeivce()
```

```
9           start_notification()
10          counter = counter + 1;
11          return
12      }
13      if(deviceNotConnected) {
14          connectDevice()
15          start_notification()
16          return
17      }
```

The norification callback only post the data received from the sensor togheter with the UNIX timestamp on ROS Topic.

The pseudocod for the module is:

```
1   CreateROSTopicPublisher("heartrate")
2   InitROS(Node("talker_heartrate"))
3   HRD = CreateObject(HEART_RATE_DEVICE)
4   start_notification(HRD, notification_callback)
5   while (TRUE) {
6       sleep(30)
7       restart(HRD)
8   }
```

Chapter 6

The News Module adaptation experiment

6.1 Experiment design and data analysis

TASK: This experiment test the adaptation of the News Module. The adaptation consists in the changing of the font size of the news.

SENSORS: In this experiment we are going to record the RGB, the thermal data, the GSR and the heart rate. From the RGB we are going to extract the AUs, and the blinking From the thermal data we are going to extract the temperature on the face of the participants We are going to extract the HEART RATE by using the physical sensor (it use the HRBSM) We are going to extract the GSR by using the physical sensor

CONDITIONS: It is going to have 6 conditions:

1. : Newspaper is read aloud by the robot (no interface shown)
2. : Newspaper is shown on the torso of the robot (no sound) and no font size adaptation
3. : Newspaper is shown on the torso of the robot (no sound) and the font size is adapted
4. : Sound and visual interface with no font size adaptation
5. : Sound and visual interface with font size adaptation
6. : 5 with moving from 2m to 60cm

This means 16 interactions per participant.

QUESTIONNAIRES: It is going to use the following questionnaires:

- MEQ -> to see if morningness/eveningness has an effect
- AASP -> to see if a person prefers visual or auditory
- EPQ -> to see if personality influences the distance where the person feels most comfortable

SCENARIO: The experiment has the following scenario: It use an armchair and a Tiago robot. The robot will invite the person to take a seat on the armchair, it will give the instructions and then the experiment can begin. The robot will show/read the news at three different distances from the person:

- 2 m -> close phase of social distance
- 1.2 m -> far phase of personal distance
- 60 cm -> close phase of personal distance

At each distance the robot will display/read one news, once it finishes reading or the participant confirms that he/she finished reading the news it will move to the next distance. Once all the distances have been covered, the robot informs the participant that the experiment is over and he/she can leave the room.

The idea of the experiment is to find out which is the best combination (distance, font size, speech/no speech) based on the sensory profile of the participant, its personality, and current internal state.

Each participant will test all conditions. In the end, we will need around 15-20 participants.

6.2 Thesis experiment implementation

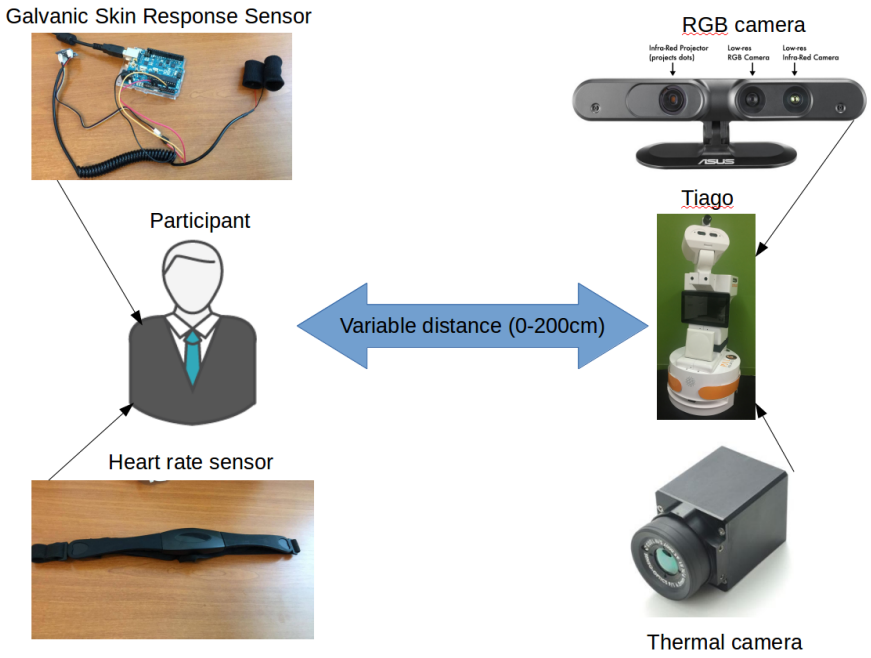

Figure 6.1: Experiment architecture

For the experiment the News Module has some difference:

1. three new css files for every one of the three distances
2. getParams - subscribe to ROS topic /distance and every time the distance change it verify in what range it is and loads the necessary css file. This adapt the interface to the distance between user and robot.

The HRBSM run on a ROS package. The heart rate sensor is place on the person. And the HRBSM use its specific address fro connectiong to it. With the help of the others module we take data like blinking, GSR, temperature of the face, and AUs.

More experiemtns are currentyly done this will be submited on ICSR in one week.

Chapter 7

Conclusion and future work

The thesis implement three modules on the project ENRICHME which are currently used. Over this it provides three frameworks which can be used by developers in other project making the implementation easier. The experiment is also an framework for testing the adaptation of your application.

The three JavaScript frameworks are:

1. lic_socket_object - a signaling API for comunicating with the backend server using a countinous connection. This is an alternative for REST API, in case you want an app more interactiv
2. lic_call_object - a video calling API over webRTC which does not contains the hard to understand architectural of webRTC. This handle inside the errors where the problems which can raise when a peer-to-peer connection is created. Also the API has few command and very easy to understand for futher developing
3. lic_news_object - a news agregator API which pass the security problems of rss and transform them into JSON format easier for developers to use without any need of conversions.

The implementation of this modules over ROS makes incrase their portability and open a field on using WebRTC protocol on robots.

Future Works The future works for the modules:

- Social Module will be to implement the WebRTC for ROS on C++ and Python, and creating an TURN server which works over proxy networks.
- News Module will be to implement an stand-alone open-source rss2json.com API, and doing some adaptaion for the categories of the news or newspaper.
- Heart Rate Bluetooth Sensor Module transform it into a more generic module, not only for heart rate, and not only for BLE device
- The experiment to be run also for the other parts of the ENRICHME project.

Appendix A

Project config files

A.1 News Module language file

```
{
    "translations" : {
        "buttons" : {
            "Previous"       : "Previous",
            "Next"           : "Next",
            "Say"            : "Read News",
            "Preferences"    : "Preferences",
            "Back"           : "Back",
            "ShowNews"       : "Show News",
            "ShowCategories" : "Show Categories"
        },
        "category" : {
            "sport"      : "sport",
            "poltics"    : "politics",
            "world"      : "world",
            "economy"    : "economy",
            "culture"    : "culture",
            "technology" : "technology"
        },
        "text" : {
            "news"       : "News",
            "catagories" : "Categories"
        },

        "details" : {
            "author"  : "Author: ",
            "title"   : "Title: ",
            "content" : "Content: "
        }
    },
    "news" : {
        "guardian" : {
            "img" : "images/news_logo/guardian.png",
            "name" : "The Guardian",
            "thumbnail" : "no"
```

```
36          },
37          "telegraph" : {
38              "img" : "images/news_logo/telegraph.png",
39              "name" : "The Telegraph",
40              "thumbnail" : "no"
41          },
42          "independent" : {
43              "img" : "images/news_logo/independent.png",
44              "name" : "The Independent",
45              "thumbnail" : "no"
46          }
47      },
48      "news_catagories" : {
49          "sport" : [
50              {
51                  "link" : "https://www.theguardian.com/uk/sport/rss",
52                  "parent" : "guardian"
53              },
54              {
55                  "link" : "http://www.telegraph.co.uk/sport/rss.xml",
56                  "parent" : "telegraph"
57              },
58              {
59                  "link" : "http://www.independent.co.uk/sport/rss",
60                  "parent" : "independent"
61              }
62          ],
63          "poltics" : [
64              {
65                  "link" : "https://www.theguardian.com/uk/rss",
66                  "parent" : "guardian"
67              },
68              {
69                  "link" : "http://www.telegraph.co.uk/news/rss.xml",
70                  "parent" : "telegraph"
71              },
72              {
73                  "link" : "http://www.independent.co.uk/news/uk/
                       politics/rss",
74                  "parent" : "independent"
75              }
76          ],
77          "world" : [
78              {
79                  "link" : "https://www.theguardian.com/world/rss",
80                  "parent" : "guardian"
81              },
82              {
83                  "link" : "http://www.telegraph.co.uk/news/rss.xml",
84                  "parent" : "telegraph"
85              },
86              {
```

```
87                  "link" : "http://www.independent.co.uk/news/world/
                        rss",
88                  "parent" : "independent"
89              }
90          ],
91          "economy" : [
92              {
93                  "link" : "https://www.theguardian.com/uk/business/
                        rss",
94                  "parent" : "guardian"
95              },
96              {
97                  "link" : "http://www.telegraph.co.uk/business/rss.
                        xml",
98                  "parent" : "telegraph"
99              },
100             {
101                 "link" : "http://www.independent.co.uk/news/business
                        /rss",
102                 "parent" : "independent"
103             }
104         ],
105         "culture" : [
106             {
107                 "link" : "https://www.theguardian.com/uk/culture/rss
                        ",
108                 "parent" : "guardian"
109             },
110             {
111                 "link" : "http://www.telegraph.co.uk/culture/rss.xml
                        ",
112                 "parent" : "telegraph"
113             },
114             {
115                 "link" : "http://www.independent.co.uk/arts-
                        entertainment/rss",
116                 "parent" : "independent"
117             }
118         ],
119         "technology" : [
120             {
121                 "link" : "https://www.theguardian.com/uk/technology/
                        rss",
122                 "parent" : "guardian"
123             },
124             {
125                 "link" : "http://www.telegraph.co.uk/science/rss.xml
                        ",
126                 "parent" : "telegraph"
127             },
128             {
129                 "link" : "http://www.independent.co.uk/life-style/
                        gadgets-and-tech/news/rss",
```

```
                "parent" : "independent"
            }
        ]
    }
}
```

Listing A.1: Great Britain - english language json file for News Module

A.2 STUN/TURN config file

```
#listening-device=eth0
listening-port=80
tls-listening-port=443
alt-listening-port=3478
alt-tls-listening-port=5349
#listening-ip=172.17.19.101
#listening-ip=10.207.21.238
#listening-ip=2607:f0d0:1002:51::4
#aux-server=172.17.19.110:33478
#aux-server=[2607:f0d0:1002:51::4]:33478
#udp-self-balance
#relay-device=eth1
#relay-ip=172.31.10.107
#relay-ip=2607:f0d0:1002:51::5
external-ip=52.88.157.249/172.31.10.107
#relay-threads=0
min-port=1025
#max-port=65535
verbose
#Verbose
#fingerprint
#lt-cred-mechch is default.
#no-auth
#use-auth-secret
#static-auth-secret=north
#server-name=blackdow.carleon.gov
#oauth
#userdb=/var/db/turndb
#psql-userdb="host=<host> dbname=<database-name> user=<database-user
    > password=<database-user-password> connect_timeout=30"
#mysql-userdb="host=<host> dbname=<database-name> user=<database-
    user> password=<database-user-password> port=<port>
    connect_timeout=<seconds>"
#mongo-userdb="mongodb://[username:password@]host1[:port1][,host2[:
    port2],...[,hostN[:portN]]][/[database][?options]]"
#redis-userdb="ip=<ip-address> dbname=<database-number> password=<
    database-user-password> port=<port> connect_timeout=<seconds>"
#redis-statsdb="ip=<ip-address> dbname=<database-number> password=<
    database-user-password> port=<port> connect_timeout=<seconds>"
#realm=www.xn--ciocrlan-o2a.ro
#check-origin-consistency
#user-quota=0
```

```
#total-quota=0
#max-bps=0
# bps-capacity=0
#no-udp
#no-tcp
#no-tls
#no-dtls
#no-udp-relay
#no-tcp-relay
#stale-nonce
cert=./cert.pem
pkey=./privkey.pem
#pkey-pwd=...
#cipher-list="DEFAULT"
#CA-file=/etc/ssh/id_rsa.cert
#ec-curve-name=prime256v1
#dh566
#dh2066
#dh-file=<DH-PEM-file-name>
#no-stdout-log
#log-file=/var/tmp/turn.log
#syslog
#simple-log
#alternate-server=1.2.3.4:5678
#alternate-server=11.22.33.44:56789
#alternate-server=5.6.7.8
#alternate-server=[2001:db8:85a3:8d3:1319:8a2e:370:7348]:3478
#tls-alternate-server=1.2.3.4:5678
#tls-alternate-server=11.22.33.44:56789
#tls-alternate-server=[2001:db8:85a3:8d3:1319:8a2e:370:7348]:3478
#stun-only
#no-stun
# rest-api-separator=:
#no-loopback-peers
#no-multicast-peers
max-allocate-timeout=500
#pidfile="/var/run/turnserver.pid"
#secure-stun
mobility
#proc-user=<user-name>
#proc-group=<group-name>
#no-cli
#cli-ip=127.0.0.1
#cli-port=5766
#cli-password=qwerty
#server-relay
#cli-max-output-sessions
#ne=[1|2|3]
#no-tlsv1
#no-tlsv1_1
#no-tlsv1_2
```

Listing A.2: STUN/TURN config file

Bibliography

[1] Roxana Madalina Agrigoroaie and Adriana Tapus. Developing a healthcare robot with personalized behaviors and social skills for the elderly. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction*, pages 589–590. IEEE Press, 2016.

[2] Oya Aran, Dairazalia Sanchez-Cortes, Minh-Tri Do, and Daniel Gatica-Perez. Anomaly detection in elderly daily behavior in ambient sensing environments. In *International Workshop on Human Behavior Understanding*, pages 51–67. Springer, 2016.

[3] Tracy A Bedrosian and Randy J Nelson. Sundowning syndrome in aging and dementia: research in mouse models. *Experimental neurology*, 243:67–73, 2013.

[4] Adam Bergkvist, Daniel C Burnett, Cullen Jennings, and Anant Narayanan. Webrtc 1.0: Real-time communication between browsers. *Working draft, W3C*, 91, 2012.

[5] RSS Advisory Board. Rss 2.0 specification (2009). *URL: http://www.rssboard.org/rss-specification*, 2014.

[6] Kevin Bouchard, Jean-Sébastien Bilodeau, Dany Fortin-Simard, Sebastien Gaboury, Bruno Bouchard, and Abdenour Bouzouane. Human activity recognition in smart homes based on passive rfid localization. In *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*, page 1. ACM, 2014.

[7] Kristina Chodorow. *MongoDB: the definitive guide*. " O'Reilly Media, Inc.", 2013.

[8] Diane J Cook, Aaron S Crandall, Brian L Thomas, and Narayanan C Krishnan. Casas: A smart home in a box. *Computer*, 46(7):62–69, 2013.

[9] Jim Downing. Web feeds and repositories. 2008.

[10] Erina Ferro, Michele Girolami, Dario Salvi, Christopher Mayer, Joe Gorman, Andrej Grguric, Roni Ram, Rubaiyat Sadat, Konstantinos M Giannoutakis, and Carsten Stocklöw. The universaal platform for aal (ambient assisted living). *Journal of Intelligent Systems*, 24(3):301–319, 2015.

[11] David Flanagan. *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", 2006.

[12] J Huiting, H Flisijn, ABJ Kokkeler, and G Smit. Exploiting phase measurements of epc gen2 rfid tags [c]//rfid-technologies and applications (rfid-ta). In *2013 IEEE International Conference on*, 2013.

[13] Salvatore Loreto and Simon Pietro Romano. *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser*. " O'Reilly Media, Inc.", 2014.

[14] R Mahy. Network working group j. rosenberg request for comments: 3489 j. weinberger category: Standards track dynamicsoft c. huitema microsoft. 2003.

[15] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun). Technical report, 2010.

[16] Mozilla Developer Network. Same-origin policy, 2014. *URL: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy (b ezocht op 10-05-2015).*

[17] S Ozses and S Ergul. Cross-domain communications with jsonp, 2009.

[18] Sandipan Pal and Charith Abhayaratne. Video-based activity level recognition for assisted living using motion features. In *Proceedings of the 9th International Conference on Distributed Smart Cameras*, pages 62–67. ACM, 2015.

[19] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[20] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L Littman. Activity recognition from accelerometer data. In *Aaai*, volume 5, pages 1541–1546, 2005.

[21] Jonathan Rosenberg. Interactive connectivity establishment (ice): A methodology for network address translator (nat) traversal for offer/answer protocols. 2010.

[22] Mohammad Mostafa Soltani. *Neighborhood Localization Method for Locating Construction Resources Based on RFID and BIM*. PhD thesis, Concordia University, 2013.

[23] Guido Van Rossum et al. Python programming language. In *USENIX Annual Technical Conference*, volume 41, page 36, 2007.