Project acronym	**CAMI**
Project number	**AAL 2014-1-087**
Project full name	**Artificially intelligent ecosystem for self-management and sustainable quality of life in AAL**
Dissemination level	Restricted
Type of deliverable	Report
Contractual Date of Delivery	M18
Actual Date of Delivery	M18
Deliverable Number	D2.3a
Deliverable Name	Report on the progress of system implementation
Work package / Task	WP2 / Task 2.3
Work package responsible / Task responsible	MDH / MDH
Number of Pages	42
Contributors	Ashalatha Kunnapilly, Cristina Seceleanu, Alexandru Sorici, Irina Mocanu, Oana Cramariuc
Version	3
Keywords	System implementation, CAMI Gateway, Decision Support System, Multi-modal user interface, Telepresence Unit
Abstract	In this report, we describe the implementation progress of the CAMI system, which is developed as a core system component with a set of programming interfaces that can be used by several modules, offering different services to the user. All the modules are developed independently, however, they provide seamless integration to the CAMI core. We give the overview of the CAMI implementation, after which we detail on the current implementation of the fall detection and alarm system, physical exercise monitoring, program management, initial human-robot integration, vocal interface, implementation of the decision support system in CAMI, under two different scenarios, one critical (cardiac arrest and fall), one non critical (reminders and daily planning). We show that CAMI's architecture presented in D2.2 ensures a modular implementation that can be smoothly integrated and extended.

Table of Contents

Table of Contents

Executive summary

Aim of the deliverable
The aim of this deliverable is to record the progress of CAMI system implementation. The implementation details of CAMI skeleton along with the individual plug-in modules are discussed along with reference to certain scenarios.
Brief description of the sections of the document
The document is structured into sections that highlight the CAMI system implementation progress. Initially an overview of CAMI architecture implementation strategy is presented by highlighting the micro-service based approach for CAMI gateway implementation. From Section 3, the individualised modules of CAMI system implementation are discussed. Section 3 describes the fall detection system by using sensor data fusion from wearable fall detection sensor and ambient sensor like camera. In Section 4, module for physical exercise management is discussed by modelling the training activity as a game with two avatars. We have dedicated a Section 6 to describe the human-robot interaction using Tiago robot and its integration to CAMI platform. As part of Section 7, the implementation details of CAMI vocal interface is described by describing certain scenarios involved that require voice processing. We have dedicated a section to CAMI DSS implementation and described the behaviour of DSS using a critical and a non-critical scenario. Section 9 gives the concluding remarks for the deliverable, followed by the references.
Major achievements
The major achievements of this deliverable are (i) recording the implementation details of CAMI system by highlighting its modularised development (ii) Knowledge of existing technologies available in market (iii) Streamlining the existing technologies to support CAMI functionalities.
Summary of the conclusions obtained
A broad description of the overall CAMI system implementation, modularised development of functionalities and scenario based analysis of components are described as part of this deliverable.

List of Figures

List of Tables

1. Introduction

CAMI is an integrated AAL architecture solution that provides support for a multitude of functionalities like services for health management, home management and wellbeing (including socialization, and reduced mobility support). A detailed description of CAMI architecture and their respective functionalities were elaborated in the deliverable D2.2.

CAMI architecture is developed as a **Core system component** with a set of APIs that can be used by several modules, offering different services to the user. All the modules are developed independently; however, they provide seamless integration to the CAMI core.

As part of this deliverable, the progress regarding the implementation of various CAMI modules are recorded. In Section 2, the overview of CAMI system implementation is presented. Section 3 deals with the implementation of fall detection system combining the data from wearable fall sensors and ambient sensors to detect a fall event of the elderly and raising an alarm to inform the caregivers and family. Following that, we have Section 4 dealing with the physical exercise monitoring and Section 5 that deals with program management. We detail the human robot integration in CAMI in Section 6, detailing the interactions of CAMI system with the Tiago robot. Section 7 deals with the vocal interface support for CAMI. In Section 8, the implementation details of CAMI Decision Support Systems (DSS) are documented by highlighting the functionality of DSS in response to a critical scenario involving a cardiac arrest and fall and non-critical scenario involving issuing reminders and daily activity planning, followed by Section 9 which deals with summary and

conclusions.

2. CAMI System Implementation: Overview

As outlined in the D2.2 deliverable on the system architecture, CAMI has six major functionality blocks: Sensor Unit, Data Collector Unit, CAMI Gateway, Multimodal User Interface, Telepresence Unit and CAMI Cloud (see also Figure 1).
This initial report on system implementation will more closely examine the block that lies at the core of the system (since it interfaces with all the other blocks): the CAMI Gateway.
Details about work on the Multi-modal user interface (vocal interface, to be specific) and the Telepresence Unit are also given.

The CAMI Gateway relies on a micro service composition approach to implement its functionality. The core of the Gateway is composed by two modules: the Event Stream Manager and the Local Store (MySQL DB). The former is essentially a message passing broker that ensures communication between the different microservices.
The latter records the short-term history of events (e.g. end-user health measurements, scheduled physical exercises, performance log of physical exercise execution, call history).
These two modules ensure that the rest of the CAMI Gateway microservices detailed in this report can receive new data as soon as it enters the system, as well as have the ability to perform operations (e.g. as is the case for the Decision Support System) on information that has been collected previously (i.e. events stored in the MySQL DB).

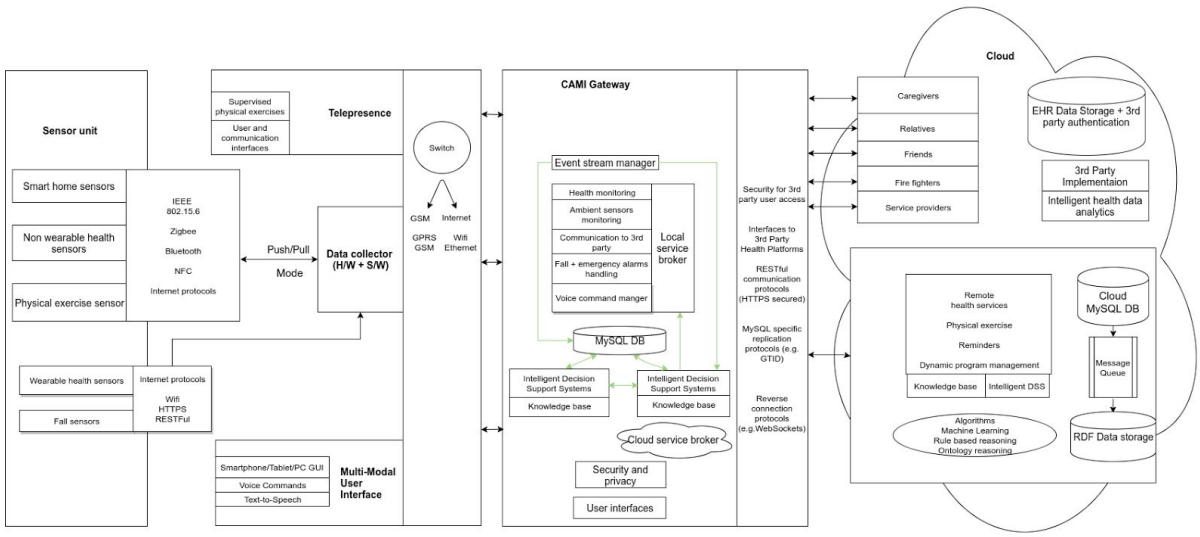

Figure 1 CAMI Block Architecture Diagram

To enable even more decoupling, but also to allow easy deployment and integration, the core modules of the CAMI Gateway, as well as the other microservices, are deployed as **Docker[1] containers.**

Figure 2 shows an example deployment of the CAMI Gateway. To keep the figure readable only the CAMI Datastore (the short term history MySQL DB), the Health Monitoring and the CAMI Data Access microservices are depicted.

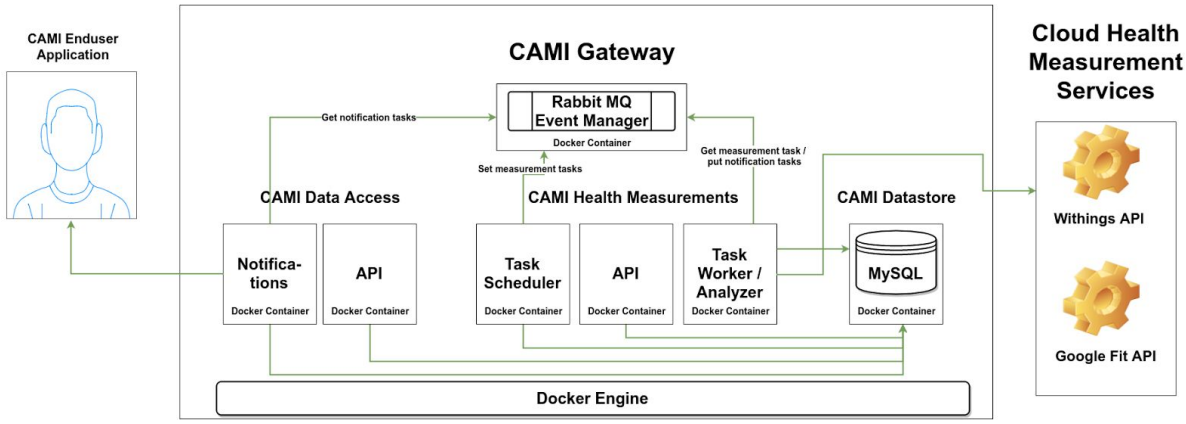

Figure 2 Dockerized CAMI Gateway deployment (showing only Health Measurement, Notification and Storage microservices).

The diagram illustrates that the services themselves are highly modularized, such that each individual component of the service can be independently customized and monitored. For example, the Health Monitoring service has three components, each deployed in its own light-weight Docker container. The first one is a Task Scheduler which manages the periodic tasks of retrieving health measurements from devices connected to the CAMI Sensor Unit, or directly from the cloud (for devices that send their data directly to a cloud service - e.g. Withings weight scale, LG smartwatch sending data to Google Fit).
The Task Scheduler inserts these tasks into the Event Manager (as notification of new available task), the Task Worker/Analyzer is triggered by the new task notification and performs the retrieval and analysis of the new measurement (possibly looking at previous measurements as well).

To retrieve data from previous measurements (e.g. for use in the analysis), both the Task Scheduler and the Task Worker/Analyzer use the API component. The latter uses the Django ORM (object relational model) to read and write from the database.
The Task Analyzer may use the Decision Support System (explained in Section 8) and determine that a notification must be sent to the end-user. In that case, a notification task is placed on the Event Manager, which is picked up by the Notifications component of the CAMI Data Access service. The latter service enables access of the client application running on a smartphone or tablet to/from the CAMI Gateway. The Notifications component of this

[1] http://docker.io

service makes use of the Push Notifications functionality for iOS/Android to send appropriate messages to the user (e.g. that he/she is doing fine, that they should try to relax - if the heart rate is too high, that they should consider a walk, if a period of inactivity has been detected).

Since there are a lot of microservices that run in the CAMI Gateway, many containers are expected in a typical deployment. To enable easy lifecycle management of all these containers, we employ a container management solution called Rancher[2]. This is both a command line, as well as a browser-based container deployment management platform which enables a developer to bring up, close, configure, monitor and maintain many Docker containers.

At the current stage of development, the core of the CAMI Gateway is implemented and running, meaning that all the other relevant microservices can be easily integrated. In the Sections that follows we explore the individual progress for each one of them.

3. Fall alarm

Falls are serious threats to elderly people living alone and can even result in life threatening situations when the fall is critical and not addressed within a specific time. As a result, in CAMI, fall detection and raising an alarm on the occurrence of a fall to inform the caregivers and family is given due importance.

The CAMI system employs two different sensors by which we can validate the occurrence of fall event of the elderly. One of them is a wearable sensor like the Tunstall fall detection sensor (readily available in the market) and the other is ambient sensor (non wearable) like a camera. It should be noted that the non-wearable sensor (camera) can be relied upon to detect the fall only when the person is inside the home premises. If the person is outdoor, the fall detection is entirely processed with the help of wearable sensor. So, we have also associated location tracking using location sensor nodes as part of this fall detection system in order to effectively determine the fall event. In what that follow, we describe the Fall detection module of CAMI assuming that the sensor data comes from the Tunstall fall detection sensor and the camera and the location monitoring sensor nodes (assuming one location sensor in each room of the home).

The **Tunstall fall detection sensor**[3] employs a two level fall detection system. The sensor wakes up from sleep when it determines a huge impact and then uses another sensor to calculate the person's orientation to determine if the person is 'on the ground'. If the person is identified to be 'on the ground', it also checks the duration for which the state continues and if it exceeds a predefined interval of time, it signals that the fall event has occurred. In order to

[2] http://rancher.com/rancher/

[3] http://www.tunstall.co.uk/solutions/ivi

determine the impact and then identify the person's state and time duration of person lying on the ground, we use a two stage fuzzy inference system (FIS) to process the sensor data.

Fuzzy logic has been proved very effective to analyse sensor data to determine various events [4]. The advantages of using fuzzy in determining events from sensor data are:

1) It can efficiently handle imprecise and unreliable sensor value readings.
2) It does not use crisp logic, i.e., if we say that the person lying on ground unattended for a time duration of 400 seconds is critical and model this as a crisp deadline, the system wouldn't even consider a duration of 399 seconds to be critical, which in reality is equally critical.
3) When compared with other classification algorithms, fuzzy logic based on linguistic variables is more intuitive and easier to use.

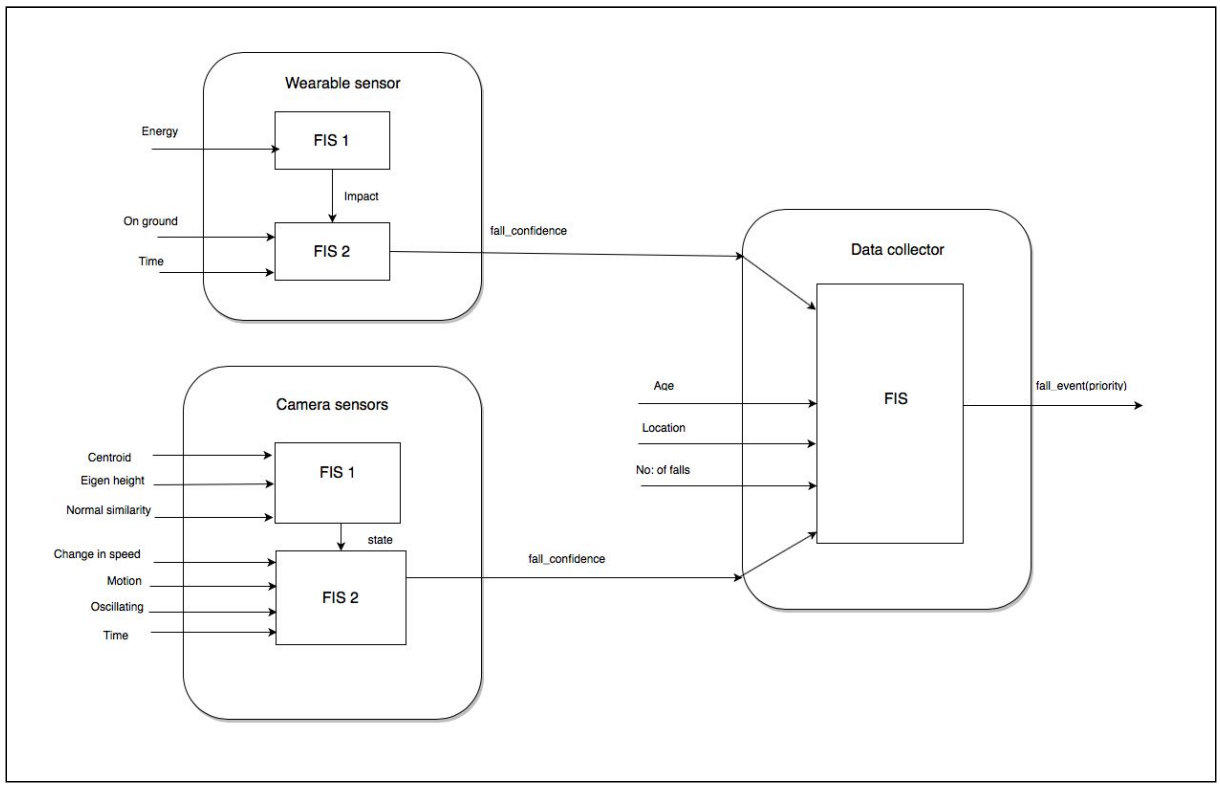

Figure 3 Fall detection system

The two stage FIS used for fall detection is depicted in Fig. 3. As shown 'signal energy' can be associated with a threshold value to effectively determine the impact of the fall using the first FIS. We have associated the signal energy with linguistic variables: Low (0% to 30%) Medium (20%-60%), High (50%-100%). The impact value produced as output is also fuzzified as Low (0% to 30%), Medium (20%-60%), High (50%-100%). This impact value can be fed as input to the second FIS, which also looks into two other input features - whether the orientation of the person shows that the person is 'on ground' {associated with fuzzy inputs Low (-0.5 -0.2 0.2 0.5) Medium = (0.1, 0.5,0.5,0.9) and High = (0.5 ,0.8,1.2,1.5) } and the 'time duration' {fuzzified as brief (-1,1,1,2), short (1, 5, 10, 15), moderate (10 120

480 720) and long (480, 900, 900 ,901); where the numbers represent time in seconds} for which the person is on ground [5]. As we have not collected any real data from fall sensors, we have fuzzified the inputs based on values in the literature wherever available and assumptions if not. The fuzzy rules of FIS 1 and FIS 2 are tabulated and shown in Table 1 and 2 respectively.

Rule		Threshold (energy level)		Impact
1	If	L	Then	L
2		M		M
3		H		H

Table 1 Fuzzy rules for FIS 1 of wearable fall sensor

Rule		Impact	On ground	Time duration		Fall
1	If	H	H	L	Then	H
2		M	H	H		H
3		L	L	B		L
4		L	L	S		L
5		M	M	M		M
6		H	M	S		M
7		H	M	B		M

Table 2 Fuzzy rules for FIS 2 of wearable fall sensor

Similarly, if the person is inside home premises, then fall event can also be monitored via multiple cameras, which are ambient sensors. For feature extraction and analysis of data from the camera, we have been dependent on the research work [5][6]. Again, we use fuzzy logic for decision making due to the obvious reasons listed above. We employ a two stage fuzzy system like that of Tunstall sensor. FIS 1 is used to identify the person's state (on ground, in between, upright fuzzified as very low = (−0.5, 0, 0, 0.5) , low = (0 0.25 0.25 0.5) , medium = (0 0.5 0.5 1) , and high= (0.5 1 1 1.5) using features like centroid, eigen height and

normal similarity extracted from voxel person, which is a 3-D representation of user generated by employing multiple cameras [5] and if the state information is 'on ground' , then we consider various other features like change in speed {Low (0, 1, 1, 1.3), Medium (0.8, 1.2, 1.2 ,1.6), High (1.5, 2, 100, 102)}, motion when the person is in ground state {Low (-0.2 ,0, 0, 0.2), High (0.1, 0.4,100,102)}, number of times the person tries to get up from on the ground state shown as oscillating behavior {Low (-2,0,2,4), Medium (1,3,5,7) and high (4,6,8,10); once the person reaches the upright position, the fall event is discarded }and time duration of person lying on the ground (fuzzified same as for wearable sensor) to detect whether a fall has occurred or not [6]. The fuzzy rules of FIS 1 and FIS 2 are tabulated and shown in Table 3 and 4 respectively and we use the same values of data inputs as specified in [5][6].

Rule		Centroid	Eigen Height	Normal Similarity		Upright	In Between	On the Ground
1		H	H	H		L	V	V
2		M	H	H		L	L	V
3		L	H	H		V	L	L
4		H	M	H		V	H	V
5		M	M	H		V	H	L
6		L	M	H		V	H	H
7		M	L	H		V	L	H
8		L	L	H		V	V	M
9		H	H	M		L	V	V
10	If	M	H	M	Then	L	L	V
11		L	H	M		L	H	V
12		H	M	M		L	H	V
13		M	M	M		L	H	V
14		L	M	M		V	H	L
15		L	M	L		V	L	H
16		L	L	M		V	L	M
17		H	H	L		H	V	V
18		M	H	L		M	V	V
19		L	H	L		L	L	V
20		H	M	L		M	L	V
21		M	M	L		L	L	V
22		L	M	L		L	H	V
23		M	L	L		V	H	L
24		L	L	L		V	L	H

Table 3 FIS1 of Ambient sensor (camera) [Reference: 5]

Rule		On the Ground	Time Duration	Change In Speed	Motion	Oscillating		Fall
1		High	Long					High
2		High	Moderate	High				High
3		High	Moderate		High			High
4		High	Moderate		Low			High
5		High	Moderate			High		High
6		High	Short	High				Medium
7	If	High	Short			High	Then	Medium
8		High	Short			Medium		Medium
9		Medium	Moderate	High	Low			High
10		Medium	Moderate		Low			High
11		Medium	Short	High				Medium
12		Medium	Short			High		High
13		Medium	Short			Medium		Medium

Table 4 FIS2 of Ambient sensor (camera) [Reference: 6]

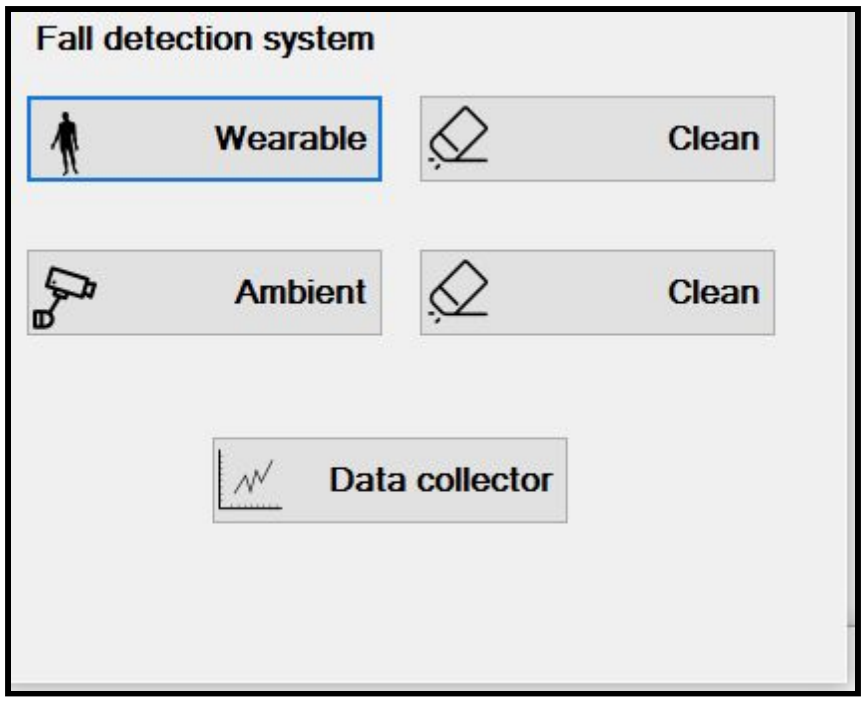

Figure 4 GUI of the Fall Detection System

The fall event detected by both the wearable and non-wearable sensors are associated with a confidence interval and is communicated to the CAMI Data collector (Fig.1). The data collector has a FIS system to decide the priority of the fall event combining the inputs from both these sensors {fall_confidence: Low (-0.5, -0.2, 0.2, 0.5), Medium (0.1, 0.5, 0.5,0.9), high (0.5,0.8,1.2,1.5)}, ,number of previous falls of user {Low (0-4) High (3-10)}, age of the user {Young(20-40 years), middle-age(35-55), old (50-100)}} and also location node data (Location_node_bedroom(Low,Medium,High),Location_node_living_room(Low,Medium,Hig

h) Location_node kitchen(Low, Medium, High), Location_node_bathroom(Low, Medium, High) where Low (0-30), Medium (20-60), High (50-100) represent signal strengths in percentages} hence chances of raising a false fall alarm is significantly reduced. The fall event eventually gets communicated to the CAMI gateway, where it gets registered in the event stream and the decision support system (DSS) takes the decision to inform the caregivers and family members if the fall is critical.

The implementation of the above logic has been successfully completed to detect the fall event.The GUI for demonstrating the fall detection scenario is shown in Fig.4 However, we have not yet considered the real data from fall sensor and camera sensor for the analysis. After extracting the relevant features from the data, we need to verify if we can still use the same fuzzy sets for the successful determination of fall event, and if it doesn't some of the features and feature values may be altered in the final system implementation.

4. Physical Exercises

An application as a serious game that aims to help elderly people to learn how to perform physical exercises in order to maintain a healthy lifestyle in their homes was developed. The application is implemented as a game with two avatars: user and trainer. The user avatar must reproduce the movements of the trainer's avatar. The user movements are captured using a Kinect sensor. A screenshot of the game is given in Figure 5.

Figure 5 Screenshot of the game

The main parts of the game are:

- animating a 3D avatar that will represent the user in the environment
- computing a distance between an exercise done by the user and the one done by the trainer
- displaying in a friendly manner the score calculated using the distance

- saving data in logs for each user to track its evolution

The game was implemented using the 3D engine Unity3D. The structure of the game is described in Figure 6. It has three main components: (i) a component that is responsible with recording the set of the reference exercises (*Capture Reference Exercises*); (ii) Set up the user profile - setting up user information: health status and recommendation (*Set up User Profile*), (iii) exergame creator and interpretation (Exergame Manager). Data are saved into *Data Repository*.

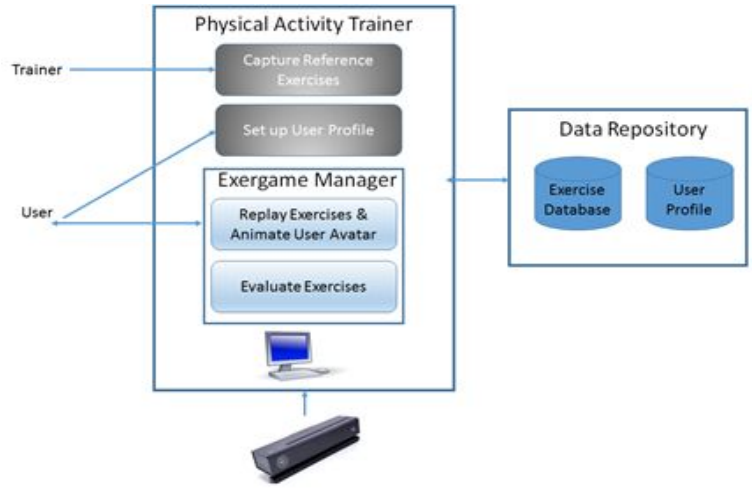

Figure 6 . Game components

Capture Reference Exercises - We used the features of the Unity 3D engine to record motion using the Kinect One sensor and then display it on the screen using a 3D avatar. First of all, we built a humanoid 3D avatar with a skeleton animation (using Blender 3D) and imported it in the graphics engine.

For each exercise we'll store in *Exercise Database* (from *Data Repository*) (i) exercise name, (ii) exercise description, (iii) exercise category, (iv) difficulty level, (v) relevant joints for the exercise, (vi) repeating time. For example – exercise rotation arms is recorded exercise description: starting position: stand up straight, your feet slightly apart and exercise routine – put your hands on the shoulders and rotate simultaneous your elbows four times forward and then four times backwards. All these information will be stored into the Exercise Database from the Data Repository. Each exercise is described as a sequence of postures. Each posture is represented by the list of quaternions associated to skeleton joints. We use the following skeleton joints: Spine, ShoulderCenter, Neck, ShoulderLeft, ElbowLeft, HandLeft, ShoulderRight, ElbowRight, HandRight, HipLeft, KneeLeft, FootLeft, HipRight, KneeRight, FootRight (as given in Figure 7). All quaternions are relative to "HipCenter" joint.

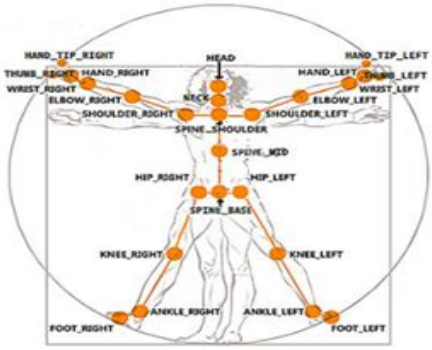

Figure 7 Skeleton joints

An exercise has the following general structure:

<frameNumber>
<bodyRootPosition> : (x, y, z)
<joint1quaternion> : (x1, y1, z1, w1)
...
<jointMquaternion> : (xM, yM, zM, wM)

There are N frames: each one contains a 3D vector (body root position) and M joints quaternions. These values will be used to determine the similarity between real-time movements (performed by the user) and pre-recorded movements (that will be performed by the avatar).

Set up User Profile will save into the *Data Repository – User Profile Database* all information associated with each user. All these data will be will be saved at the first time when an user will start to play the game. He will fill in: his medical information (what is his health status, what are recommendations regarding the physical activity that he can perform, for example he must make physical exercises with arms, legs, trunk and the difficulty level that is accepted for his profile). All these information will be set up through a questionnaire in which the user must select one option.

The *Exergame Manager* is responsible with both animation of the trainer and user avatars and also with computing the score associated to the performed exercise. The trainer avatar is controlled through the exercises recorded into the *Exercise Database*. The user avatar is updated real time through the movements realized by the user. These movements are captured through the skeleton provided by the Kinect sensor and mapped to the avatar using the Unity package - Kinect Wrapper Package for Unity. These information are updated through the Replay Exercise module. For user movements we use 21 joints from the skeleton provided by the Kinect sensor. All these joints are transmitted through the Unity 3D engine and are mapped 1-to-1. Animator class from Unity is used for facilitating the mapping between the

standard Kinect skeleton's joints and the 3D model's joints, as given in Figure 8.

Figure 8 . a) Avatar's skeleton joints. b) Automatic mapping between standard joints and avatar's joint

However, the skeleton tracking is slightly noisy. It is important that the data sent to the avatar be smooth, and, to achieve this, Holt-Winters Kalekar, double exponential smoothing filters are used. The method is also devised under the name of second-order exponential smoothing, which is the recursive application of an exponential filter twice.
In other terms, exponential smoothing is a filtering method that acts similarly to low-pass filters, removing high frequency noise (jitter). In the project, this technique is implemented as a standalone script that also includes multiple levels of corrections (smoothness) that can be set by the user.
Evaluate exercise computes the final score associated with an exercise. The distance between user and trainer movements can be calculated using DTW (Dynamic Time Warping) algorithm because this technique aligns the gestures so that the most appropriate postures are compared. DTW has good results for comparing two gestures. The DTW module receives as input two exercises: user's and a reference, populates the frame data structure and calls the compare method for each repetition:

```
DTW(user, ref) {
        user_data = getData(user);
        ref_data = getData(ref);

        for each rep in user data:
                current_res = compare(user_rep, ref);
                current_path = path(user_rep, ref);
}
```

The compare method finds the best match between the two series and updates the distance matrix. Also a path is computed for each repetition, therefore, at this point, for N repetitions, the algorithms outputs N results that must be aggregated into one single score.
Score computing: in order to compute the score associated to an exercise the following steps are made:

1. Compare two quaternions q1 and q2: the unit quaternions unit_q1 and unit_q2 are computed, the distance dq between them is computed using:
   dq = 1 - |inner(unit_q1, unit_q2)|
   where inner is the scalar product between two vectors. This metric returns a number between [0, 1] and relies on the fact that if two quaternions are very similar, their inner product is 1 (thus, the result of the comparison is 0), and that if two quaternions are very different (orthogonal), their inner product is 0 (thus, the result of the comparison is 1).
2. Compare 2 frames f1 and f2: the distance between two frames (df) is computed as the mean squared error distances between pairs of quaternion (dq) associated to joints from both frame f1 and f2;
3. Compute the total score associated to an exercise as the mean square error of distances between pairs of frames (df).

5. Program Management

The Program Management Microservice comprises a set of functionalities ranging from the typical inspection of a calendar of daily events to the intelligent recommendation of activities or thoughtful rescheduling of postponed/missed events (e.g. rescheduling a physical exercise or a health measurement, taking into account future activities).

At the current stage of implementation, we have focused on building the common expected program management functionality, that is the ability to view/create/modify activities grouped into four categories:

- medication reminders
- physical exercises
- health measurement requests
- personal (i.e. other end-user activities such as visits to friends, various appointments, etc)

For each type of activity, the user has the option of defining either a one-time event, or a recurrent one. When defining a recurrent activity, the user can specify:

- an event that recurs several times a day (e.g. daily at 08:00 and 18:00)
- an event that recurs several times a week, specifying the given weekdays (e.g. every Monday, Wednesday and Friday at 12:00)

The results of end-user focus groups reported in D1.5 showed a of end-users to have the possibility for synchronization and sharing of their activity schedules with other existing tie management solutions (e.g. Google Calendar, iCalendar).
The current implementation allows also for a synchronization of the local calendar with a Google Calendar instance that belongs to the end user.
Furthermore, the user can directly define activities in a CAMI specific calendar of their Google Account, which gets automatically synchronized with the CAMI Gateway.
In the Google Calendar, each one of the four activity types has a different color code to make

it easy for the end-user to distinguish between them.

As explained in Section 2, the program management service is implemented in its own Docker container. Its components resemble those of the Health Monitoring microservice.
The synchronization operation between local calendar and a Google Calendar instance is therefore performed similarly. A Synchronization Task component handles recurrent syncing.
An API component allows access to/from the local calendar for an application client (e.g. the CAMI mobile or tablet interface).
A Program Analysis Component works with the Decision Support System to determine if a recommendation or a notification based on the current created/modified activity needs to be issued.
The latter intelligent analysis is deferred for the second

6. Human-Robot Interaction

6.1 Tiago

6.1.1 Overview of Tiago robot

Tiago is manufactured by the Spanish company PAL Robotics with the aim of being a research and development robot that adapts and fits to a wide range of research applications (Figure 9) [7].

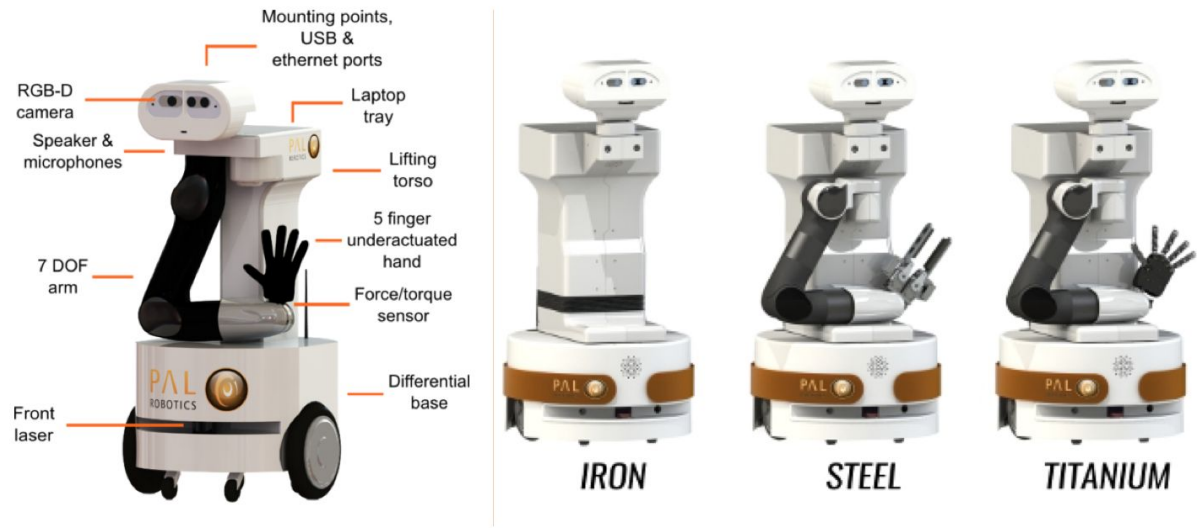

Figure 9 Tiago configuration (left); Iron, Steel and Titanium versions of Tiago.

Tiago has manipulation capabilities in its Steel and Titanium versions which comprise

a extensible robot arm that ends in a gripper (Steel) or a hand (Titanium). Its size, sensing and manipulative capabilities make Tiago suitable for the type of assistive scenarios envisioned in the CAMI project. Table 5 is presenting Tiago's general and technical specifications in the Steel configuration purchased by CITST.

General Features	Height	110 - 145 cm
	Weight	70 Kg
	Footprint	ø 54 cm
Degrees of Freedom	Torso lift	1
	Mobile base	2
	Head	2
	Arm	7
	Total (without end-effector)	12
Steel Configuration	Mobile Base	Yes
	Navigation Laser	5.50 m
	Lifting Torso	Yes
	Pan-tilt head	Yes
	7 DoF arm	Yes
	End-effector	Parallel Gripper
	Force/torque sensor	Yes
Body technical specs	Arm payload (at full extension)	2 Kg
	Arm reach (without end-effector)	87 cm

	Torso lift	35 cm
Mobile base technical specs	Differential drive	Yes
	Max speed	1 m/s
	Operation environment	Indoor
Connectivity	Wireless connectivity	802.11 n/ac 2x2 Dual Band Wi-Fi
	Bluetooth connectivity	Bluetooth 4.0
Electrical features	Battery 36V 20Ah	2 batteries
	Battery Autonomy	6h - 10h
Sensors	Base	Laser 5.5m or 10m range, rear sonars 3x1m range
	IMU (Base)	6 DoF
	Motors	Actuators current feedback
	Torso	Stereo microphones
	Head	RGB-D camera
Computer	CPU	Intel i7 Haswell
	RAM	16 GB
	Hard Drive	256 GB SSD
Software	OS	Ubuntu Linux LTS

	Open source middleware	ROS
	Periodic updates/patches	Yes
	Arm with	position / velocity / effort control
	Eye-hand calibration suit	Yes

Table 5 Tiago general and Steel specific hardware configuration and capabilities.

6.1.2 Robot Operating System (ROS)

The Tiago middleware is Robot Operating System (ROS)[8] . ROS is a collection of software frameworks for robot software development providing operating system-like functionality on a heterogeneous computer cluster. It is a meta-operating system by design, offering hardware abstraction, low-level device control, message-passing between processes and package management. It also provides tools and libraries for communication between different computers or robots running ROS. This collection of tools, libraries, and conventions aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms. This extensive compatibility is also one of the main strengths of the framework.

Because ROS is an open source project aimed to support multiple robots and boards, there is a plethora of open-source modules for robot software development. It offers a flexible framework for writing robot software and encourages collaborative robotics software development. The ROS community is composed of 15.000 active users, with more than 1.4 million page views this year.

Running as a distributed framework of processes, called Nodes, it enables loosely coupled nodes to communicate using message passing. Each node can be included in a package or stack which can be shared and distributed through the ROS packaging system. Processes in ROS are communicating using the “Computation Graph”, a peer-to-peer network. The basic concepts of the Computation Graph are nodes, Master, Parameter Server, messages, services, topics and bags.

The nodes in ROS are usually processes which enable communication with certain modules like lasers, rangers, RGB and depth cameras and usually send data or receive commands from the Master node. The Master process provides name registration and lookup for the rest of the Computation Graph and basically routes the data in the graph. The parameter server allows data to be stored in a central location and it is currently implemented inside the Master node. Messages are a data structure composed by a set of primitives and array of primitive types (integer, floating point, boolean, char).

6.1.3 Integration with the CAMI platform

Current integration with the CAMI platform is done through the EXYS9200-SNG Enhanced Data Collector which developed by the EXYS partner organization as part of the CAMI box (see also D2.4a). A bidirectional link between the EXYS9200-SNG and the CAMI Gateway

via a RESTfull / JSON mechanism allows full communication among the CAMI Robot and the CAMI Gateway. The EXYS collector (see Figure 10) is a high scalable and modular platform which has been augmented for the CAMI project with the openHAB (v.2) framework and the software binding with ROS enabling full communication between openHAB and any CAMI ROS-based robot. At the same time, the openHAB iot-bridge[11] plugin is establishing a bidirectional communication with IoT systems thus enabling the collector to talk with both ROS and CAMI integrated home-automation sensors.

Figure 10 The EXYS9200-SNG Data Collector.

Installation of the iot-bridge (ROS binding for openHAB) by EXYS was performed as follows (see D2.4a for further details). Supposing that the ROS source directory on the Robot is ~/catkin_ws/src, open a terminal and run the following commands:

```
cd ~/catkin_ws/src
git clone https://github.com/corb555/iot_bridge.git
cd ..
catkin_make
```

- Edit ~/catkin_ws/src/iot_bridge/config/items.yaml file by inserting the IP and port of the openHAB
- Edit openHAB's item file
  - Create the ROS group: Group ROS (All)
  - Add the (ROS) group to each item that should send status updates to ROS.

6.1.4 OpenHAB

OpenHAB[11] is software for integrating different home automation systems and technologies into one single solution that allows over-arching automation rules, and that offers uniform user interfaces. OpenHAB 2 is an open-source solution based on the Eclipse SmartHome framework. It is fully written in Java and uses Apache Karaf together with

Eclipse Equinox as an OSGi[4] runtime and bundles this with Jetty as an HTTP server. It is highly modular software, which means that the base installation (the "runtime") can be extended through different kinds of "add-ons", either to communicate with new home automation solution, or to offer a new kind of user interface. The structure of openHAB is summarized in Figure 11.

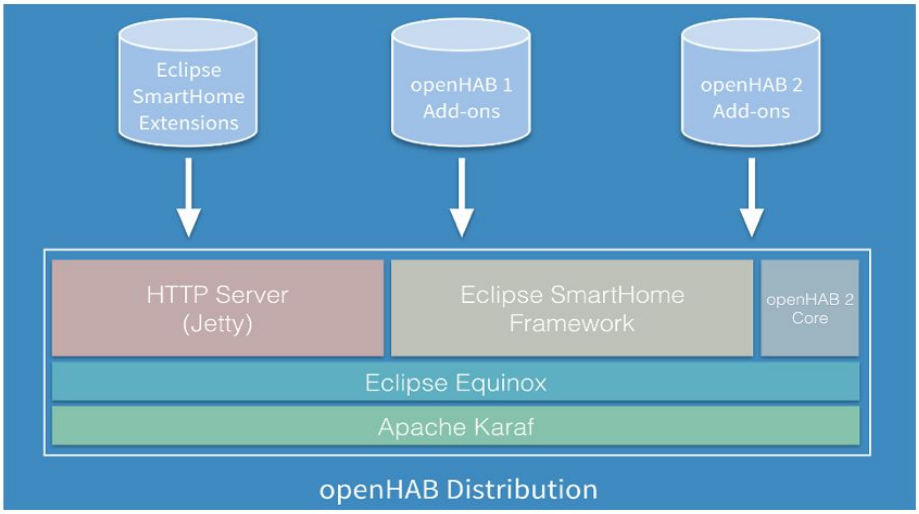

Figure 11 OpenHAB 2 structure

For the CAMI system, OpenHAB 2 was adapted and installed on the EXYS9200-SNG Enhanced data collector, managing the Home Automation network of sensors and the communications with CAMI Robot (ROS). In particular, its services are:

- a Web GUI allowing the operator to configure the HA sensors system and the Robot
- a Web GUI allowing the user to interact with the HA sensors system and the Robot
- an API layer for communicating via REST/JSON with the exterior (the CAMI Gateway, in our case)

6.1.5 Scenarios

Two simple scenarios have been envisaged for a first robot demonstration session of the CAMI project. These scenarios involve the usage of OpenHAB through the CAMI interface which runs on a tablet, communication between OpenHAB and Tiago, navigation of Tiago. UML diagrams of the scenarios are presented in D2.4a.

1) First scenario
Technical description: (1) get status of light switch; (2) turn off light.
Story: Tiago goes to the user with the tablet either because the user calls Tiago (verbal,

[4] https://www.osgi.org/

gesture) or because Tiago wants to inform the user about information received from the home automation sensors integrated with OpenHAB. The user checks on the tablet the lights in a remote location (e.g. his countryside house) and realizes that some lamps are ON. He switches them OFF using the OpenHAB interface.

2) Second scenario
Technical description: (1) get status of window; (2) Tiago navigates to the window; (3) Tiago closes window.
Story: Tiago is at home and the user at a remote location. The user checks on the openHAB interface if any window is forgotten open at home. He finds one window open and presses the close window button. Then the robot will go to the window and will close it. The first implementation of this scenario will not involve autonomous navigation of Tiago which is currently being tested and developed (See section 6.1.6). A simulation is prepared (using a PC with ROS installed acting as a Robot), in order to demonstrate the communication between the Robot and the CAMI Box. Tiago will be manipulated remotely using a console.

6.1.6 Setting up Tiago navigation framework

This section presents the autonomous navigation framework of Tiago and initial simulation performed for the implementation of the scenarios described in section 6.1.5. The ROS 2D navigation stack is the basis of TIAGo navigation. The navigation software is composed of all the ROS nodes running in the robot that are able to perform SLAM, i.e. mapping and localization using the laser on the mobile base, and path planning to bring the robot to any map location avoiding obstacles and preventing collisions using laser and sonars readings [12] The architecture of thee Tiago navigation software is shown in Figure 12.

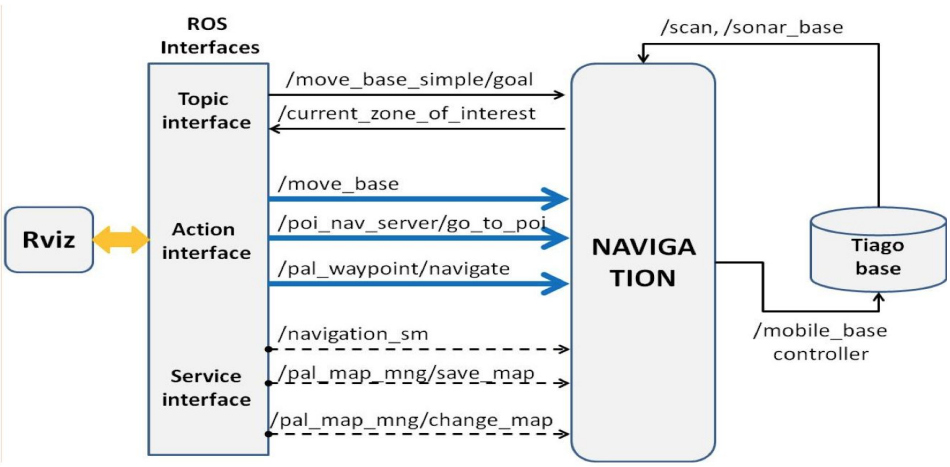

Figure 12 Tiago navigation software architecture

.

The user can communicate with the navigation software through a ROS topic, three different

ROS actions and ROS services.

/move_base_simple/goal: Topic interface to send the robot to a pose specified in /map metric coordinates. This is used if no monitoring of the navigation status is required.

/current_zone_of_interest: This topic prints the name of the region of interest where Tiago is at present, if any.

/move_base: Action to send the robot to a pose specified in /map metric coordinates. It is recommended when the user needs to get noticed when the goal has been reached or whether something fails in the process.

/poi_navigation_server/go_to_poi: Action to send the robot to an existing Point Of Interest (POI) by providing its identifier. POIs can be set using the Map Editor which comes with Tiago.

/pal_waypoint/navigate: Action to make the robot visit all the POIs of a given group or just a subset of them.

/pal_navigation_sm: Service to set the navigation mode to mapping or to localization mode. In the localization mode the robot is able to plan paths to any valid point of the map.

/pal_map_manager/change_map: Service to choose the active map.

Finally we test Tiago navigation on different experiments in the Tiago simulation environment provided by PAL Robotics. In our simulations we work in crowded environments. Once the motion planner finds that an object is impeding another, updates the information and calls the task planner. The task planner calls again the solver with the new states. Iteratively the same process is repeated until the motion planner executes a valid plan. Results of the simulations are shown in Figures 13,14. In Figure 5 Tiago is in localization and path planning mode. The Rviz window shows the map, the localization particles, the robot model localized, the laser sensor readings and some virtual obstacles.

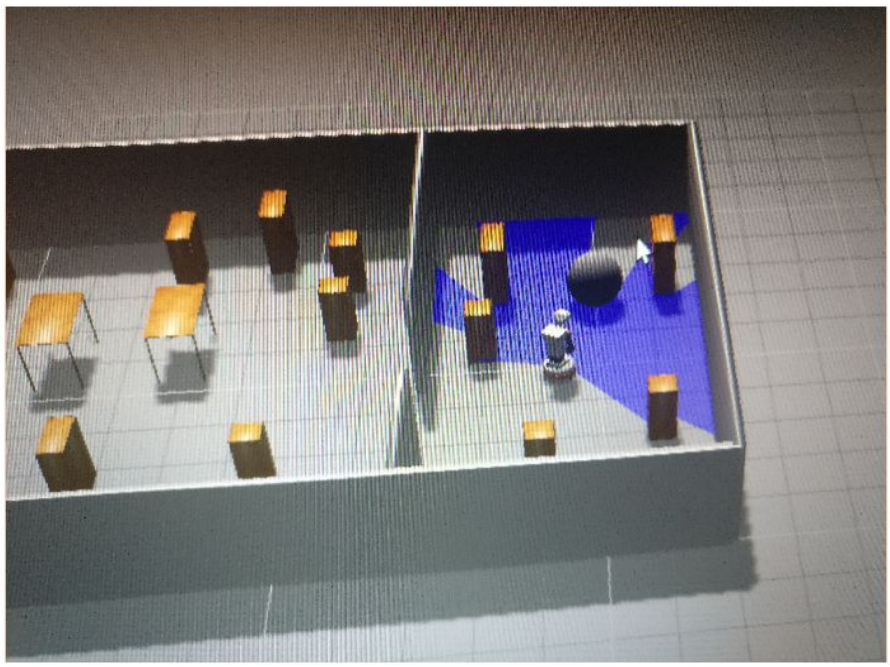

Figure 13 Office environment obtained after the simulation in localization and mapping mode.

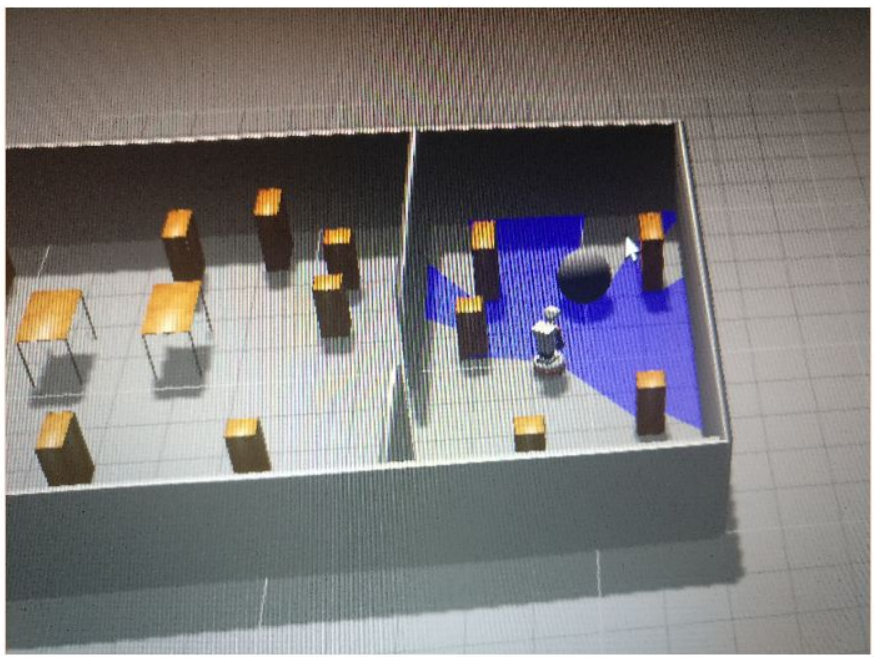

Figure 14 The Gazebo window showing Tiago in an office like environment with dynamic objects represented by the moving sphere.

7. Vocal Interface

The Vocal Interface allows vocal interactions between the CAMI system and its users. The user has access to different components of the systems through speech commands along with the capability of the system to generate vocal outputs. The Vocal Interface is composed of five main parts: Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), Dialog Management module (DM), Natural Language Generation (NLG) and Text to Speech synthesis (TTS). The flow of the interface is illustrated in fig. 15.

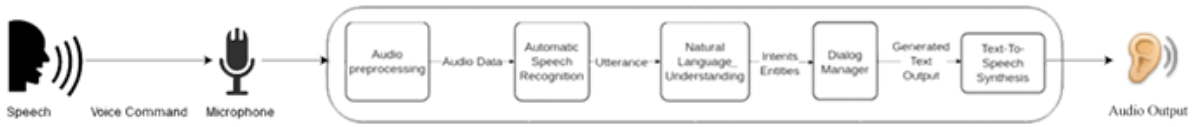

Figure 15 The flow of the Vocal Interface

Audio pre-processing is done to ensure the best results for the ASR submodule. This includes setting the proper frequency, number of channels and bit depth. The ASR module is the very important as it enables a system to detect and use speech as a method of interaction. Using a limited grammar based speech recognition system will lead to a lower user experience. An example of limited grammar speech recognition system is CMU Sphinx. It is an open-source speech recognition system first developed by Carnegie Mellon University in 2000. It has been tested with different configurations and increasing the grammar to provide a better user experience was decreasing the success of the recognition. The response time however is very good since CMU Sphinx was developed for low-resource platforms. Speech recognition APIs are now broadly used. Some examples of such APIs are Google Speech API, Wit.ai and Microsoft Bing Speech Recognition API. They all offer good response time while maintaining a high success rate of recognition. Most of them offer a query limited free to use subscription and SDKs for speech ending detection. The ability of interacting using spoken language requires less effort for communication and improves the user experience. Although speech recognition is a complex task, the natural language understanding and the dialog manager both play a big part in enabling interactions with the system.

Natural language processing is necessary for the speech interaction by offering language understanding support. The system receives a plain text utterance, determine the intent of the user and extracting the relevant entities, then outputs it to the Dialog System. The simplest way of processing the phrase is by using regex expressions to extract the relevant information. This incurs a limitation in user expressivity, an overhead in the matching system and low flexibility to changes in the training data. Ensuring a better user experience requires a more complex natural language processing of the utterance. The Wit.ai API was trained with AAL

and AMI domain specific queries. This is a free to use framework for natural language understanding. It classifies the training data into intents. An example of such intent is Reminders, where all the data related to reminders should be classified. The API also finds key entities labeled in each sentence. For example the sentence Remind me to call Mary at 12 PM Today the JSON received from Wit.ai should classify it as "*Reminders*" and recognize key entities like *Reminder=call Mary, Time=12 PM Today*. The intents used in this work and their description are described in Figure 16.

The API allows the use of entities which are restricted to certain values or more general ones which are recognized from the sentence as a whole. For example for the intent *MedicalFactsInformation* the entities accepted in this intent are *MedicalFact*, datetime and number. The *MedicalFact* entity can only take values between *Glycaemia*, *Blood pressure* and *Pulse*. The system can also use spanless entities, which do not require labelling them in the training phase and are recognized from the whole phrase. For example, the phrase "*The question is difficult*" and "*Can you give me more difficult questions*" the difficulty entity could match the word difficult but with different meanings difficulty=low and difficulty=high. The dialog system will be triggered by the intent of the utterance and will match the entities so that it can correctly process the phrase. The entities can also have a role attached. For example the intent Medical Facts Information can classify sentences containing information about blood pressure. The blood pressure values require two number values, systolic and diastolic. To identify the query I used two roles attached to the number entity, primary and secondary. Roles can be also used for queries like "*I want to buy a ticket from Bucharest to London*" could be used to describe the entity location as *destination = London* and *origin = Bucharest* have an intent called movies which it is used for movie playing. The API will not be able to correctly differentiate between the movie title and the director. For example, the query "*Play me The Godfather from Francis Coppola*" and the query "*Show me movie directed by Peter Jackson called Lord of the Rings*", the API can only assume that the movie title will be first and the director last, which will be wrong in this example. Wit.ai API allows the use of text or audio queries. The response of the API is a JSON a list of possible outcomes. An outcome contains the intent and a list of entities, as well as a confidence score which is the probability of correct intent classification. The API offers a list of special, prede_ned entities like datetime, duration and numbers. This allows the translation of text queries like "*this Friday*" to the corresponding UTC timestamp or "*for twenty minutes*" as a duration in seconds.

Intent	Description	Examples
Environment Interactions	This intent is used for AmI interactions.	switching on/off the lights, turning on/off the air conditioning, raising/lowering the blinds
Medical Facts Information	Used for adding new information regarding medical facts.	Heart rate, blood pressure, blood sugar level information and queries
Medical Facts Queries	Used for queries regarding medical facts.	
Medication Information	Used for adding new information about medical treatments.	Inserting new information about the date, time and number of pills taken.
Medication Queries	Used for queries regarding the medical treatment.	Asking information about date, time and number of pills taken.
Physical Exercising Information	Used for adding new information about physical exercising.	Adding information about time and duration of the work out.
Physical Exercising Queries	Used for receiving information about physical exercising.	Asking the system about previously added physical exercising information.
Repeat	Used for asking the system to repeat something.	Please repeat that
Social Interactions	Used for understanding general social interactions.	Hello,goodbye, see you later.
Reminders Informations	Used for telling the system new reminders.	Reminders can contain anything the user needs to later on be reminded about.
Reminders Queries	Used for asking the system about reminders.	The user can ask about any reminders in a period of time that he previously added.

Figure 16 Intents and entities

The Dialog Manager is the core of the speech interaction. Its structure can be seen in Figure 17. It is responsible with extracting the relevant data after matching the processed utterance and offer a response to the user. Once the intent is determined by the natural language processing module, the system should be able to di_erentiate between a query, in which it should gather the necessary data and provide a text output to the user and/or to the *Text-To-Speech Synthesis* module, a command in which it should trigger the AMI System Interface and data insertion in which the system should save the relevant data about the user. The Dialog manager's main tasks are choosing what action to perform based on the input received from the NLU system and extract the relevant information from the AMI system, while maintaining history information about the dialog.

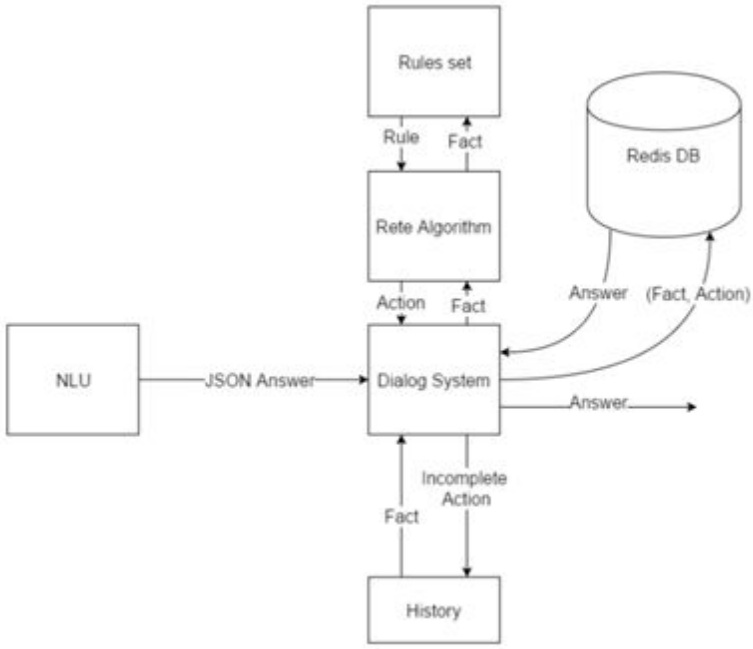

Figure 17 The Structure of the Dialog Manager

The system is implemented as a rule based approach. The labelled sentence received from the NLU contains the intent and a list of key-value pairs. The phrases that the dialog system should understand are carefully represented so that key entities can be extracted and matched as rules. For example, the phrase" *Did I take my pills today?"* will be matched as a *MedicationQuery* since it asks information about the medication schedule and the key entities extracted are ("action" : "*take*"), ("*medication*" : "*pills*") and ("*datetime*" : "*today*"). The rule on which the dialog system should match this utterance can be seen in the following equation (1):

equals(intent; MedicationQuery) ^ contains(medication) ^ contains(action) ^ contains(date) (1)

Any utterance classified as a *MedicationQuery* which contains a medication, an action and a date entity will be matched against this rule. The values of the entities are not important for the matching system. The matching system sole responsibility is to find the action to perform. The matching system uses a Forward Chaining Algorithm for matching the rule. The algorithm is suitable as it implements a production rule system which matches the fact against a set of predefined rules. The JSON received from the NLU system is transformed into a fact for the input of the algorithm. The algorithm tries to find a match for the fact. It contains two types of rules: complete and incomplete rules. The complete rules are the ones that once matched it can provide an answer to the user directly. The incomplete rules are the ones that

require additional information from the user to offer a complete answer. For example, the equation

equals(intent; MedicationQuery) ^ contains(medication) ^ contains(action) (2)

is one of the incomplete rules of the equation (1), because it misses the date for offering a complete answer to the user.

The algorithm will still match the rule, yet the action chosen as a result will set the missing entities in the answer. In the example above, the action matched by the rule in equation (2) - Incomplete rule will insert the date entity as a missing entity. The *Dialog manager* will check for the missing entities in the Dialog History. If it can find a previously asked entity it will add the missing predicate in the fact and retry the matching process. If not, it will ask the user to offer the missing entities and will set the context to ensure that only the missing entities will be matched. The system will ask the user for the missing date. The NLU system will not be able to classify every intent possible so an answer like ”*Today*.” From the user will not be classi_ed correctly by the NLU yet the entities will be extracted nevertheless. The Dialog manager needs to save the missing entities in the context and check if something different from the default context has been set. If so the Dialog manager will add the entities received to the fact saved in the context and run the matching algorithm again. Now the fact will contain the missing date and will be matched against a complete rule and the user will receive a proper answer. If the user offers a different answer, for example asks the system to turn on the light, the Dialog manager won't find the missing entities in the answer from the NLU, yet will find a complete rule to match the new query against. The dialog manager will then change the context to the default one, match the rule, offer a complete answer to the user and then ask for the missing entities once again. If the user chooses to ignore this a second time the dialog manager will drop the context permanently. The context of speech is implemented as a node containing a different rule set, a fact. The dialog manager always checks if a context is different from the default set. If so, the system combines the two facts and be checked against the rule set in the context. If no match can be done, the dialog system looks into the history to find the missing entities. If no match is still made it moves back to the default context. If the contexts time to live index is zero, the context is permanently destroyed.

After matching the rule, the correct action is chosen. The actions to perform for equation (1) is to query the Redis database with all the medications matching the date, that have been taken by the user. Each action is implemented as a POST query to request to the defined endpoints. The body of the request contains the key elements in the fact for the query. This way the *Dialog System* can be easily extended with new rules and new actions by adding the rule in

the rule set and implementing the end point. The history is saving the list of the last 5 facts passed to the Forward Chaining Algorithm. This way the dialog manager can process queries for repeating information and undoing certain queries.

The following example contains a query in which either the ASR module had error in correctly recognizing the user utterance or the user made a mistake:
1 User: I have exercised for two hours today at 5 pm.
2 System: You exercised for two hours today at 5 PM. I saved the information.
3 User: I did not say that!

These errors should be able to be reversed by the dialog manager. The first utterance will insert the information of exercising for a duration of two hours starting from 5 pm in the database. The system will output the information to the user so that he can disconfirm if the system didn't get the information properly. The last utterance ”I didn't say that!” represent a disconfirm from the user and will be matched as an undo action. The undo action will take the last fact from the history and match it against the rule system again, but this time telling the system to invert the operation. In this case, the system will erase the information of exercising for two hours starting from 5 pm from the database.

Text-To-Speech synthesis converts the generated output of the dialog manager module to audio. This is especially used for short messages as "*I didn't understand that*" or "*Please repeat that*" for telling the user the system failed to understand what he said or to confirm the information understood by the system. For example, sentences like "*Ok. You've took one pill at 9 am on 05.06.2016*." or "*Ok. I will remind you to call John at 6 pm*." will be the answers to queries like "*I took one pill at 9 am on the Fifth of June 2016.*" and "*Please remind me to call John at 6 pm.*". The TTS module is also used for error handling by asking the user additional information regarding the entities needed for a matching a complete rule like in the following example:

1 User: I have exercised today at 5 pm.
2 System: Please tell me the duration of the work out.
3 User: 30 minutes.
4 System: Ok. You worked out for 30 minutes today at 5 pm.

In case the user will not answer with the required entity, the TTS module will be used to inform the user the system required something different to complete the old dialog, while showing the newly requested information. For example:

1 User: I have exercised for 20 minutes.
2 System: Please tell me when you have worked out.
3 User: I've taken two pills today at 7 pm.
4 System:Ok. You took two pills today at 7 pm. Remember to tell me the date and time of the work out.

when the user tells the system something different than expected, it will receive an answer containing the asked information, while the system also reminds him to fill in the missing entity.

The user can tell the system "*Today at 9 am.*" and it will fill the context and offer a confirmation like "*Ok. You've exercised for 20 minutes today at 9 am.*", or it can choose to ignore it and the system will not ask for the missing entity anymore.

8. Decision Support System

The CAMI Decision Support System (DSS) employs the recent trends of development in the fields of AI. Since the CAMI architecture employs both local and cloud based processing, the DSS is present locally in the CAMI gateway as well as in cloud system. The local DSS is responsible for taking decisions during the occurrence of critical events like fall, cardiac arrest, health parameter variations, fire etc. which have hard real-time deadlines associated with their resolution. The cloud based DSS system is used to take decisions when the events are soft real-time, for instance, planning of daily activities of user, issuing reminders, advices etc.

The local DSS is a compositional approach of using rule-based expert system with case based reasoning systems (for critical decision making). The reason for this choice is that local DSS needs to handle all critical scenarios which are well-defined and hence defining knowledge in terms of rules gives modularity and uniformity across the problem domain. Moreover, the rules are expressed in a natural format which gives more clarity. However, the major disadvantage of rule-based expert system is that they do not work well with situations that demand common-sense and also that the system cannot be able to deduce a conclusion if there are no rules written for it. Hence, using Case Based Reasoning (CBR) systems which use analogical reasoning; i.e., by matching and adapting to a case that have successfully solved before (more similarity with human reasoning techniques) is needed to ensure the success of our DSS. For CAMI, we plan to employ a RBR first strategy for combining both these systems, i.e., only if there is no rule for a particular case, it redirects to the CBR system for solutions. However, it is the cloud DSS, which deals with non-critical scenarios.
In the following sections, we elaborate the implementation details of 2 scenarios-one critical and one non critical using the CAMI DSS.

Scenario 1 description: Fall + ECG data variations leading to cardiac arrest (Critical event: Expert system reasoning)

- Jim has unusual ECG variations detected by the heart monitoring sensor and has a heart attack and falls.
- CAMI receives input from fall detection sensor and heart monitoring sensor about fall and ECG variations.
- CAMI DSS should be able to analyse the situation and derive a conclusion that fall may be likely due to cardiac arrest.
- It then informs both of Jim's caregivers and family about the occurrence of both the events

Scenario 2 description: Issuing reminders for Jim (Non-critical)

- The cloud DSS should send reminders to Jim if he has forgotten to do any of scheduled activities, like missing a medication or reminding of an appointment with doctor etc.

8.1 Scenario 1 implementation

The implementation of Scenario 1 requires the data collector of CAMI establishing the occurrences of the events like fall and cardiac arrest reported by the corresponding sensors. We have already described in Section 3 regarding the detection of a fall event, we describe here how a cardiac arrest is determined using the ECG data from the sensors.

The ECG data represents the graphic recording or display of the time variant voltages produced by the myocardium during the cardiac cycle []. A normal ECG cycle is displayed in Fig. 2. The ECG data has many features which can be established to determine if there are any occurrences of a cardiac arrest. The features are described below.

- The ECG signal is characterized by five peaks and valleys labelled by the letters P, Q, R, S, T as shown in Fig. 18.

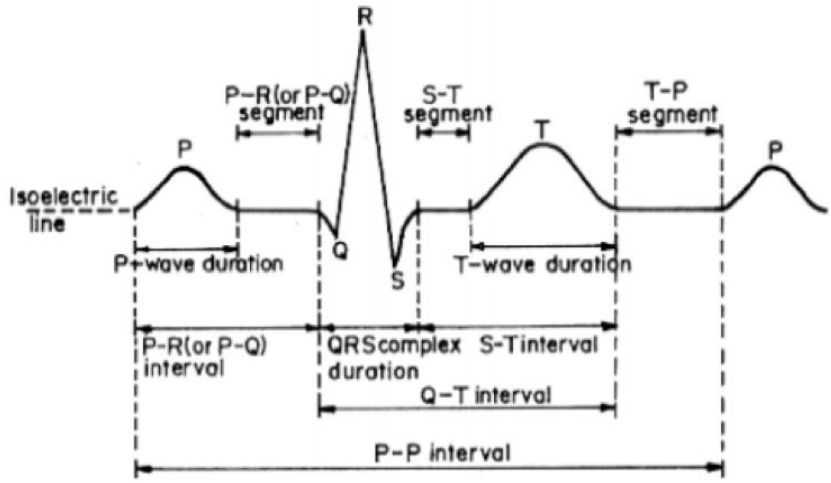

Figure 18 ECG data cycle

- In the normal state of the heart, the P-R interval is in the range of 0.12 to 0.2 seconds, the QRS interval is from 0.04 to 0.12 seconds and the Q-T interval is less than 0.42 seconds.
- The normal rate of the heart is from 60 to 100 beats per minute. A slower rate than this is called bradycardia (slow heart rate) and a higher rate is called tachycardia (fast heart rate). If the cycles are not evenly spaced, an arrhythmia may be indicated. If the P-R interval is greater than 0.2 seconds, it may suggest blockage of the AV node.

Any deviations from these normal range of data values may indicate a potential chance of a cardiac arrest and like the fall event detection, we have employed fuzzy logic to identify the event 'cardiac arrest'. The fuzzified features from ECG data are PR interval (Low, Normal, High), QT interval (Normal, Prolonged), QRS interval (Low, Normal, High), Heartbeat rate (Low, Normal, High). The fuzzy rules for detecting the cardiac arrest are summarized in Table 5. The output of the FIS in the heart monitoring sensor node can pass this information to the Data collector, which has an FIS to analyse the age, location and previous heart attacks if any and establish the occurrence of the event with a priority. This event is pushed to the event stream manager of CAMI Gateway, from where the DSS analyses the event and takes a decision.

Rules		PR interval	QT interval	QRS interval	Heart beat rate		Cardiac arrest
1		L or H	or (P)	or (L or H)	or (L or H)		M
2		N	N	N	N		L
3		L or H	P	or (L or H or N)	or (L or H or N)		H

4	If	L or H	or (N or P)	L or H	or (L or H or N)	then	H
5		L or H	or (N or P)	or (L or H or N)	L or H		H
6		or (L or H or N)	P	L or H	or (L or H or N)		H
7		or (L or H or N)	P	or (L or H or N)	L or H		H
8		or (L or H or N)	or (N or P)	L or H	L or H		H
9		L or H	P	L or H	or (L or H or N)		H
10		L or H	or (N or P)	L or H	L or H		H
11		or (L or H or N)	P	L or H	L or H		H
12		L or H	P	or (L or H or N)	L or H		H
13		L or H	P	L or H	L or H		H

Table 6 Fuzzy rules for determining ECG variations (that may lead to cardiac arrest)

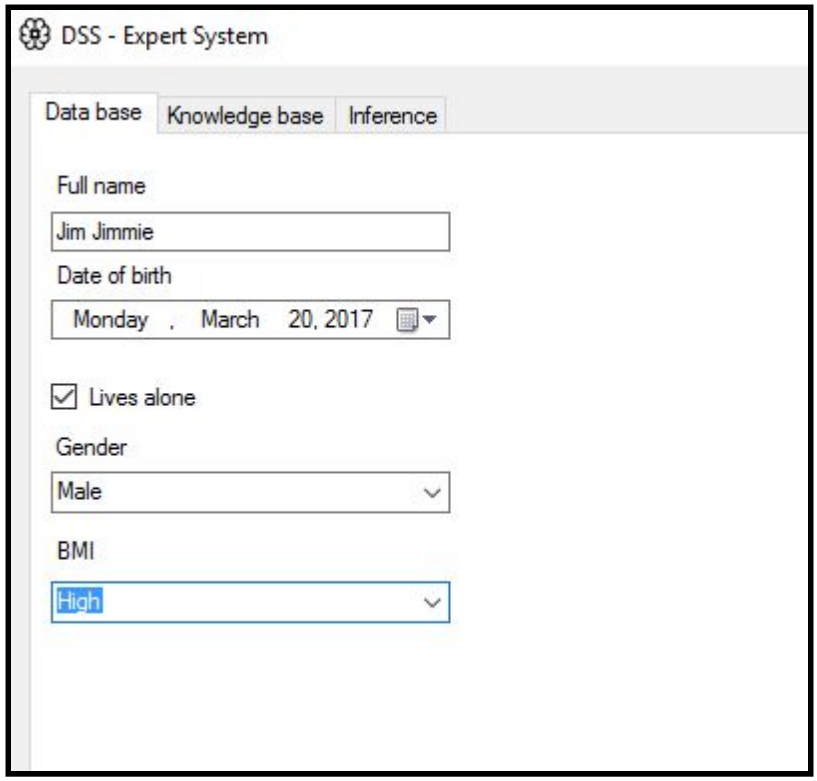

Figure 19 GUI for Expert System

8.1.1 Expert system implementation

The first task of the ES will be to determine if the events are critical or noncritical and tag them. In case of critical events, the decisions to be taken on the occurrences of such events can be pre-coded. If the events are non-critical, the ES redirects it to the DSS system in the cloud. The ES developed has a DB for storing facts and attributes, a KB for storing knowledge in terms of if-then rules and an inference engine which applies the rules in KB along with facts and attributes in DB continuously to make a decision.

The DB stores attributes related to the person Jim. Some examples include the facts like: Jim is of age 65 years, Jim's gender is male, Jim has a medical history of being a cardiac patient and subsequent falls, Jim lives alone etc. It also stores information of the critical events that can occur. At present we only consider 3 events, i.e, fall, cardiac arrest, fall due to cardiac arrest. It also stores the potential actions that need to performed. For instance, the actions taken are described below:

- Tag the events 'very critical', 'likely to be critical', 'critical', 'non-critical'.
- In case of events 'very critical', 'likely to be critical', 'critical', inform caregivers and family members.
- In case of events 'non-critical', redirect the events to the cloud system.

In the KB, we have the rule base developed by incorporating the facts in DB. The rules are of the form:

IF *condition* THEN *conclusion* AND *action*

Rules for ES

1) If event occurred is (cardiac arrest), and if the medical history indicates that the person is a (cardiac patient) and if the person (lives alone), then the event is identified to be (cardiac arrest-very critical) and (inform the caregivers and family of the event)
2) If event occurred is (fall) and the medical history indicates (subsequent falls) and if the (person lives alone), then event is identified to be a (fall- very critical) and inform caregivers and family of the event.
3) If the event occurred is (fall) and the medical history indicates (no health issues) and if the person (lives alone), then event is identified to be (fall-likely to be critical) and (inform caregivers and family of the event.)
4) If the event occurred is (cardiac arrest) and the medical history indicates (no health issues), then the event is identified to be (cardiac arrest-likely to be critical) and (inform caregivers and family of the event.)
5) If the events occurred is (cardiac arrest and fall) and/or the medical history indicates (cardiac patient and/or subsequent falls), then the event is identified to be (fall due to cardiac arrest –very critical) and inform the caregivers and family of the events.
6) If the events occurred is (cardiac arrest and fall) and/or the medical history indicates (no health issues), then the event is identified to be (fall due to cardiac arrest –very critical) and (inform the caregivers and family of the events).

The GUI developed for ES is shown in Figure 19.

8.2 Scenario 2 implementation

We emphasize the fact that non-critical events like ‘reminders’, ‘generating plans for activities’ for the user should be taken care by cloud DSS. In order to detect the events and tag them and for associating criticality of the events, we extend the expert system developed before for Scenario 1, with the difference that upon detecting a non critical event, the ES should activate the cloud DSS to solve the situation.

The following rules are added to ES in addition to the rules mentioned above by assuming that it takes inputs from calendar, user, caregiver etc.

1) If person is (Jim) and Jim’s calendar input (reminder) or caregiver input (reminder), then the event is (issuing reminders) and tag it as (‘Reminders’) and (associate low criticality to the event) and (activate cloud DSS system).

2) If person is (Jim) and Jim’s calendar input (planning activity) or caregiver input

(plans), then event is (planning) and tag it as (‘Planning’) and (associate low criticality to the event) and (activate cloud DSS system).

8.2.1 Cloud DSS system for sending the reminders.

At the current stage of implementation, we have interfaced the system with google calendar, such that the system reads the inputs from the calendar to generate the reminder event. The reminders so generated should be sent to the mobile phone carried by the user, to voice command manager and can be displayed via the telepresence. The system also considers sending the reminders based on priority. For example, if there are two events of medication reminder and reminder for taking a medicine at the same time, the system checks for priority of events (which can be defined offline) and then issues the high priority one initially and pushes aside the reminder for non-priority events like TV shows for 5 minutes ahead. Also, in case of reminders like medication, we have designed the system to give 2 repetitive reminders in the duration of 10 minutes and a provision for the user to send an acknowledgement for taking the medicine. If the system gets the acknowledgement, no further reminders are issued. Also, even after the issue of 2 reminders, the user does not give an acknowledgement, then a note is sent to the caregiver and family member that the user has missed to take the following medication. The GUI developed for the reminder generation is shown in Fig 20.

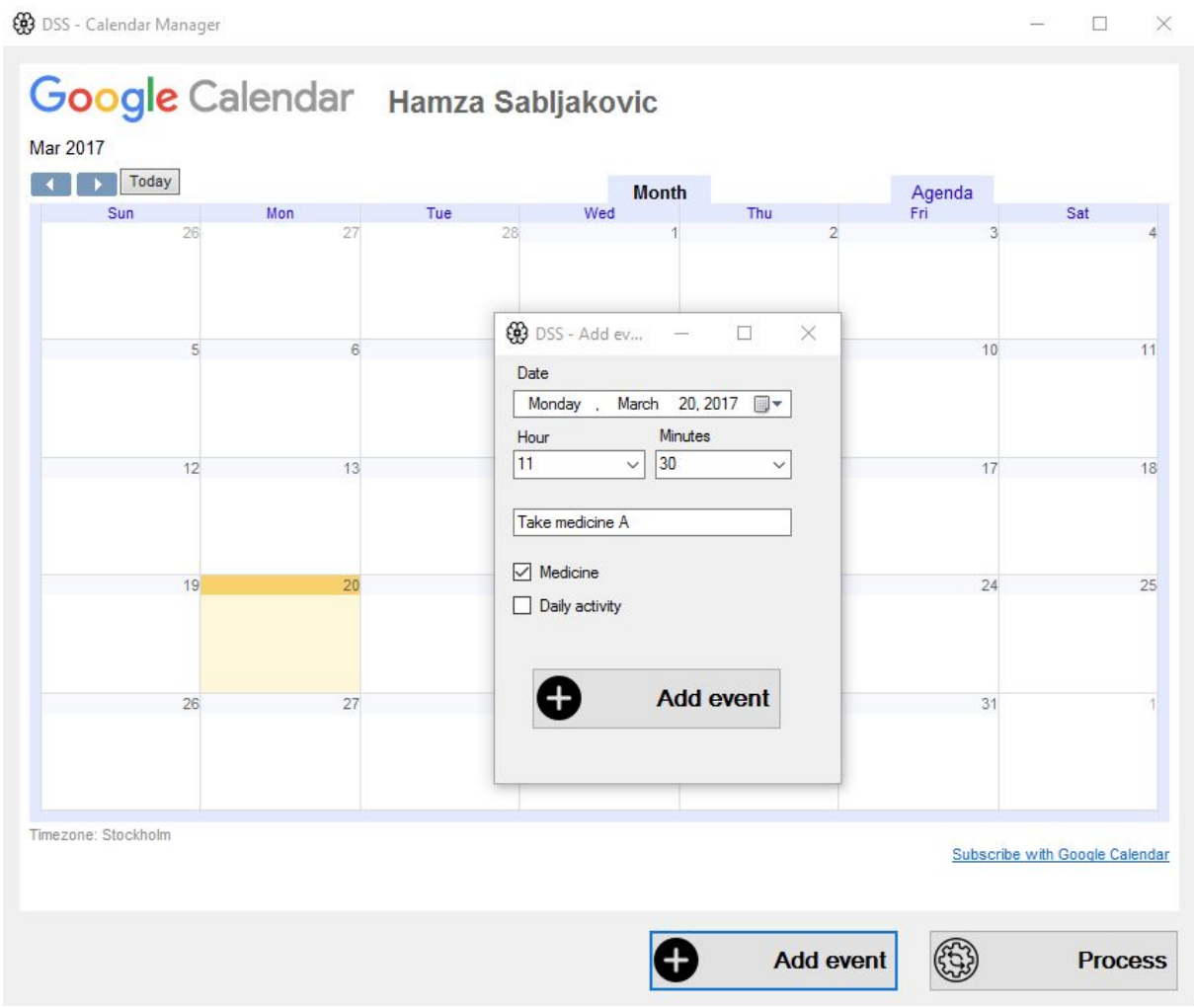

Figure 20 GUI for reminder generation by interfacing it with google calendar

9. Summary and Conclusions

CAMI architecture is a fully integrated AAL architecture comprising of various modules like Sensor Unit, Data Collector Unit, CAMI Gateway, Multimodal User Interface, Telepresence Unit and CAMI Cloud taking care of variety of functionalities like health parameter monitoring, fall detection, communication and social interaction, physical exercise monitoring, issuing reminders etc.

As already detailed in deliverable D2.2, CAMI system is based on a clean and robust skeleton, onto which several plugin modules (microservices) can be coupled.

The skeleton consists of the standard, well known technologies. The rest of the components and services (e.g. activity recommendation service, intelligent reminders, fall alarms, voice commands, exercise analysis) act as plugin components that can be linked to the skeleton framework.

As part of the current deliverable, the implementation details of various CAMI modules and the progress of system implementation are recorded.

References

1) Docker: http://docker.io

2) Rancher: http://rancher.com/rancher/

3) Tunstall fall detection: http://www.tunstall.co.uk/solutions/ivi

4) Yager, R.R. and Zadeh, L.A. eds., 2012. *An introduction to fuzzy logic applications in intelligent systems* (Vol. 165). Springer Science & Business Media.

5) Anderson, D., Luke, R.H., Keller, J.M., Skubic, M., Rantz, M.J. and Aud, M.A., 2009. Modeling human activity from voxel person using fuzzy logic. *IEEE Transactions on Fuzzy Systems*, *17*(1), pp.39-49.

6) Anderson, D., Luke, R.H., Keller, J.M., Skubic, M., Rantz, M. and Aud, M., 2009. Linguistic summarization of video for fall detection using voxel person and fuzzy logic. *Computer vision and image understanding*, *113*(1), pp.80-89

7) Tiago robot: http://tiago.pal-robotics.com/

8) ROS: http://www.ros.org/

9) Robot Operating System https://en.wikipedia.org/wiki/Robot_Operating_System

10) ROS.org http://wiki.ros.org/iot_bridge

11) openHAB https://www.openhab.org/

12) Tiago Hanbook, Pal Robotics, version 1.4.2