www.fit4work-aal.eu

# FIT4WORK

SELF-MANAGEMENT OF PHYSICAL AND MENTAL FITNESS OF OLDER WORKERS

CO-FUNDED BY

AAL PROGRAMME

The National Centre for Research and Development

GOBIERNO DE ESPAÑA MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

ZonMw

uefiscdi EXECUTIVE AGENCY FOR HIGHER EDUCATION, RESEARCH, DEVELOPMENT AND INNOVATION FUNDING

REPUBLIC OF SLOVENIA MINISTRY OF HIGHER EDUCATION, SCIENCE AND TECHNOLOGY

PARTNERS

PSNC

AWF Poznan University of Physical Education

"Jožef Stefan" Institute

UNIE KBO

SGS

teamnet transforming technology

OSM

# Knowledge Base prototype

*Ambient Assisted Living Joint Programme project no. AAL-2013-6-060*

*Deliverable 3.4, version 1.0*

| | |
|---|---|
| Lead author: | Mitja Luštrek, Jožef Stefan Institute |
| Co-authors: | Božidara Cvetković, Jožef Stefan Institute |
| | Robert Szeklicki, Poznań University of Technology |
| | Placer Vieco, SGS Tecnos SA |
| | Martin Gjoreski, Jožef Stefan Institute |

Published on 25th of April, 2016

The Fit4Work project is co-financed though the AAL Joint Programme by:

- European Commission
- National Centre for Research and Development, Poland
- Ministry of Industry, Energy and Tourism, Spain
- Executive Agency for Higher Education, Research Development and Innovation Funding, Romania
- Ministry of Higher Education, Science and Technology, Slovenia
- The Netherlands Organisation for Health Research and Development (ZonMW), The Netherlands

## Table of contents

# 1. Introduction

The Fit4Work knowledge base includes knowledge about the physical activity, the stress relief exercises, functional fitness exercises and requirements for better quality of the environment and actions which should be taken to keep the quality environment.

The physical activity knowledge base includes knowledge about most common activities and requirements on amount of daily and weekly physical activity. The knowledge base on stress relief exercises is composed of list of exercises. The knowledge about the environment quality is more complex, thus implemented and encoded into the ontology.

The knowledge base will be constantly updated and upgraded with additional information and exercises.

## 2. Physical activity knowledge base

Knowledge about physical activity is collected from two sources. The knowledge about physical activity used in the Fit4Work system is composed of knowledge about the most common activities in elderly and recommendations about the amount of physical activities per day and week.

The most common activities were retrieved from the questioner presented in deliverable on user requirements. The knowledge about the most common activities were used to compose a scenario of activities to be used for data collection. In this deliverable we present a summary of the activities and emphasise which were suitable to include in the scenario.

The most common activities are:

- Walking (23%) – most common in Spain (37%) – included into the scenario
- Gardening (18%) - most common in Romania (49%) – included into the scenario
- Cycling (17%) - most common in Netherlands (28%) –included into the scenario
- Gym/fitness (7%) – most common in Spain (16%) – included into the scenario as Fit4Work exercise
- Swimming (6%) – not feasible to include into the scenario (no water resistant sensors)
- Hiking (5%) - most common in Romania (15%) - it was included into the scenario as walking (uphill)
- Running (2%) - included into the scenario
- Sports (Football/volleyball) (2%) – not feasible to include into the scenario (multiple people needed)
- Nordic walking (2%) – included into the scenario
- Pilates (2%) – not explicitly included into the scenario
- Skiing (2%) – not feasible to include into the scenario
- Tennis (2%) – not included into the scenario
- Dancing (1%) – not explicitly included into the scenario
- Yoga (1%) – not explicitly included into the scenario

Results about most common activities was used to build a scenario to be used for data collection of activity and energy expenditure of the elderly presented in Section 2.1.

The recommendations are divided into daily recommendations and weekly recommendations. Daily recommendations include amount of active calories burned per day, at least 10 minutes of continuous moderate activity and minute of active movement per hour for at least 12 hours per day. Weekly recommendations are adapted from the WHO recommendations which state that person should be engaged into moderate-intensity physical activity for at least 150 minutes per week or vigorous-intensity physical activity for at least 75 minutes per week or comparable combination of both. More about the recommendations is reported in Deliverable 3.3.1/3.3.2.

### 2.1. Activity monitoring scenario

Data collection was done in the laboratory environment at the Faculty of Physical Education, Sport and Rehabilitation in Poznan University of Physical Education, Poland, under the supervision of physiology and sports experts. The dataset contains data of ten healthy volunteers: six male and four female, aged from 51 to 66 (59 ± 4.6) with different fitness levels, BMI from 22 to 29 (25.8 ± 2.3). All volunteers refrained from eating and drinking (except for water) in the 12 h prior the experiment. The scenario is presented in Table 2.1.

Table 2.1 Activity monitoring scenario

| | *Activities* | *Time* | *Avg MET* |
|---|---|---|---|
| ***Lying*** | Lying left side | 1′ | 1.2 |
| | Lying front | 1′ | 1.5 |
| | Lying right side | 1′ | 1.4 |
| | Lying back | 7′ | 1.2 |
| ***Basic activities*** | Walking slowly | 10″ | 3.5 |
| | Sitting down at the desk | | |
| | Sitting still | 4′ | 1.2 |
| | Sitting doing light activities (reading, writing, leafing through a book, using computer, knitting, Rubik's cube, playing cards) | 4′ | 1.2 |
| | Standing up | | |
| | Walking slowly | 10″ | 3.5 |
| | Standing still | 4′ | 1.3 |
| | Standing talking & gesticulating | 2′ | 1.7 |
| | Walking slowly | 10″ | 3.5 |
| | Standing washing hands | 2′ | 2.3 |
| | Walking slowly | 10″ | 3.5 |
| | Home chores (cooking, serving food, washing dishes, sweeping floor, washing windows) | 6′ | 2.5 |
| ***Eating*** | Eating with cutlery | 2′ | 1.9 |
| | Eating with hands | 2′ | 1.5 |
| ***Gardening*** | Planting seedlings, digging, raking, weeding | 6′ | 2.2 |
| | Rest | 3′ | |
| ***Walking*** | Walking slowly (4 km/h) | 4′ | 3.5 |
| | Walking normally (6 km/h) | 4′ | 4.2 |
| | Rest | 3′ | |
| ***Nordic walking*** | Walking normally (6 km/h) | 6′ | 4.5 |
| | Rest | 3′ | |
| ***Walking carrying a burden*** | Walking slowly (4 km/h) | 6′ | 4.2 |
| | Rest | 3′ | |
| ***Walking uphill*** | Walking slowly (3 km/h) | 6′ | 4.4 |
| | Rest | 3′ | |
| ***Running*** | Running normally (8 km/h) | 6′ | 7.1 |
| | Rest | 3′ | |
| ***Stationary cycling*** | Cycling lightly (60W) | 6′ | 4.2 |
| | Cycling normally (100W) | 6′ | 5.0 |
| | Rest | 3′ | |

# 3. Stress Relief Knowledge Base

The Fit4Work system will in addition to measuring and monitoring stress provide the user with the stress relief exercises which are a part of the stress relief knowledge base. Currently we have two types of exercises: the breathing exercises and the progressive muscle relaxation exercises. Each exercise is accompanied with the visual representation.

We will update the knowledge base with additional exercises in the future work.

## 3.1. Breathing exercises

The breathing exercises are composed of the first phase - training and the second phase - performing exercises. While training, the user should be in the comfortable position and comfortable environment without any distractive stimuli. The user should train all six breathing exercises up to the point where they can be performed anytime and anywhere. When the user masters the exercises he/she selects the exercise which suits him/her best. The exercises are presented in Table 3.1.

Table 3.1. List of current breathing exercise for stress relief which are in the knowledge base.

| Breathing exercise | Instruction | | Duration | Repetition |
|---|---|---|---|---|
| First exercise | Closed eyes | | 3 minutes | 4 times |
| | Hand under navel | | | |
| | Conduct the air to the lower part of abdomen | **Breathing** | | |
| | Imagine blowing up a balloon. | | | |
| Second exercise | Closed eyes | | 3 minutes | 4 times |
| | One hand under navel, second on the stomach | | | |
| | Conduct the air to the lower part of abdomen | **Breathing** | | |
| | Conduct the air to the stomach | | | |
| | Imagine blowing up a balloon. | | | |
| Third exercise | Closed eyes | | 3 minutes | 4 times |
| | Conduct the air to the lower part of abdomen | **Breathing** | | |
| | Exhale | | | |
| | Conduct the air to the stomach | | | |
| | Exhale | | | |
| | Conduct the air to the chest | | | |
| | Exhale | | | |
| Fourth exercise | Closed eyes | | 3 minutes | 4 times |
| | Conduct the air to the lower part of abdomen | **Breathing** | | |
| | Exhale with sonorous sound | | | |
| | Conduct the air to the stomach | | | |
| | Exhale with sonorous sound | | | |
| | Conduct the air to the chest | | | |
| | Exhale with sonorous sound | | | |
| Fifth exercise | Closed eyes | | 3 minutes | 4 times |
| | Breathing normally | **Breathing** | | |
| | Conduct the air to the lower part of abdomen | | | |

| | | | | |
|---|---|---|---|---|
| | Exhale | | | |
| | Conduct the air to the stomach | | | |
| | Exhale | | | |
| | Conduct the air to the chest | | | |
| | Exhale | | | |

In the future work we will detect which exercise is the best for the current user in the first stage from the answers retrieved from the questioner and in the second from the response retrieved from the biosensors. The most suitable exercise, which is the one that has the best impact on the stress relief will be recommended to the user.

## 3.2. Progressive muscle relaxation exercises

The progressive muscle relaxation exercises are performed for 10 to 15 minutes. The user squeezes the muscle for ten seconds and then slowly relaxes it. The exercise is composed of three phases: training phase, mental review phase and mental relaxation phase.

In the training phase, the user performs the progressive muscle relaxation exercises. The user chooses the exercise and does multiple exercises related to different body parts in predefined order for three times. All exercise are performed sitting down. The exercises are presented in Table 3.2.

Table 3.2. Progressive muscle relaxation exercises.

| Exercise | Exercise order | Instruction |
|---|---|---|
| Face, neck and shoulders training | Forehead | Wrinkle the forehead for ten seconds and relax |
| | Eyes | Open eyes widely for ten seconds and close slowly |
| | Nose | Wrinkle the nose for ten seconds and relax |
| | Mouth | Smile widely for ten seconds and relax slowly |
| | Tongue | Press tongue against palate for ten seconds and relax slowly |
| | Jaw | Clench teeth strongly for ten seconds and relax slowly |
| | Lips | Wrinkle lips (kiss) for ten seconds and relax slowly |
| | Neck | Flex neck forward for five seconds, get into initial position and flex neck backwards for five seconds, get into initial position |
| | Shoulders | Raise shoulders for ten seconds and relax slowly |
| Hands and arms training | Fist and arm | Close fist and tighten it for ten seconds. The tension is in the arm, forehand and hand and relax slowly. Repeat with other arm. |
| Legs training | Leg | Stretch leg and tighten it by pointing toes upwards for ten seconds. The tension is in the gluteus, thigh, knee, calf and foot. Repeat with the other leg. |
| Thorax, abdomen and lumbar training | Back | Arms are put across the chest in form of a cross. The elbows are pushed backwards for ten seconds. The tension is in the lower part of back and shoulders |
| | Thorax | Inhale and keep air in lungs for ten seconds. The tension is in chest, exhale slowly |
| | Stomach | Tighten stomach for ten seconds and relax slowly |
| | Waist | Tighten gluteus and thighs for ten seconds and relax |

In the future work additional exercises might be added into the knowledge base.

# 4. Ambient Conditions Knowledge Base

Ambient conditions knowledge base contains the knowledge about the parameters which are monitored, quality interval and actions which impact the parameter values. The knowledge base is encoded into the ontology.

The ontology is encoded with the OWL (Ontology web language) language, which is computer readable language to encode and express ontologies. Ontology is a set of axioms that defines object relations and relations between object and their properties. OWL can be combined with a reasoner, which (i) insures the consistency of all relations in knowledge representation and (ii) allows us to infer from implicit to explicit knowledge (e.g. sister of my father is my aunt). The first enables us to validate the ontology and second to infer current state of the ontology and properly act to the given circumstances.

The ontology was built with open-source software Protégé [10], from which we will also use some screenshots to present work done in Fit4Work. Protégé is implemented in Java language on top of the OWL API [22,9]. OWL API is Java API for manipulating and creating OWL ontologies. We utilized the Pellet reasoner [11], which can be combined with SPARQL-DL queries [5, 6, 7]. Output of the ontology is the list of feasible actions which can improve the ambient parameters in near future. The SPARQL-DL query is used to find such actions.

## 4.1. Classes and individuals

The ontology is composed of classes and subclasses. Subclasses inherit properties of their parents. OWL has superclass *Thing*, which is superclass of all classes. In OWL, there is also a class that is a subclass to all classes, called *Nothing*. In Fit4Work ontology main classes are *Building*, *Device*, *Action*, *Parameter*, *ParameterQuality* and *State*. Complete hierarchy of the classes can be seen in Figure 4.1.

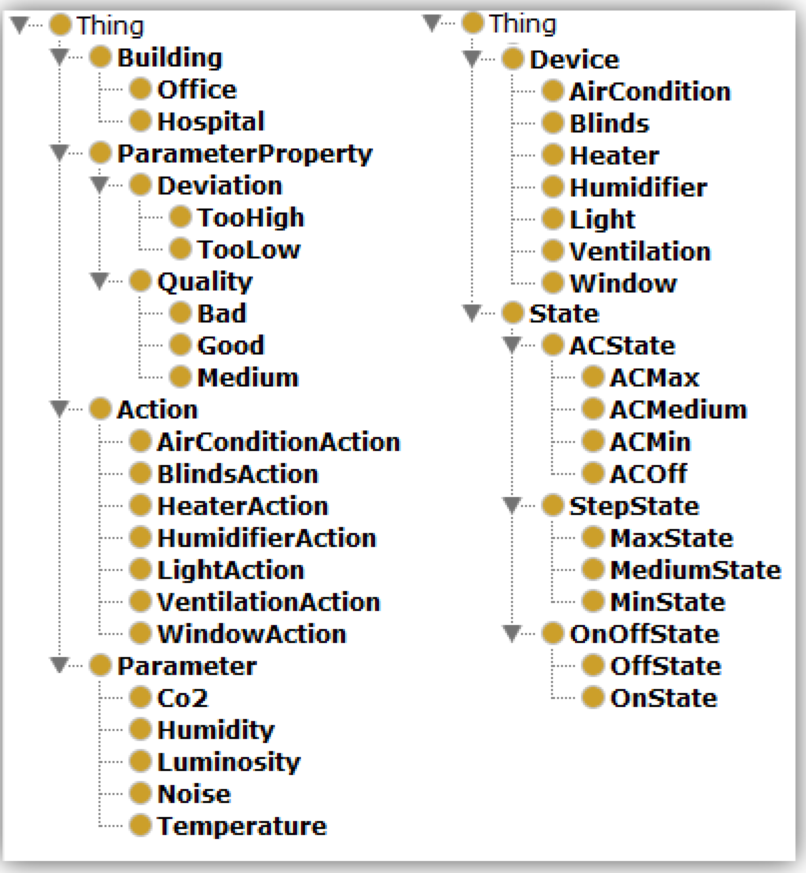

Figure 4.1 Snapshot of full hierarchy of ontology from ontology editor Protégé

The term Individuals are used for instances that can be in none, one or more classes: e.g. individual *Temperature* will be in class *Parameter* and according to its current value it will also have certain quality, therefore will be in appropriate class of *ParameterProperty*. Individual of device will be in class *Device*, but will also be in one of the *State* classes.

Subclasses in main classes, except *ParameterQuality*, are pairwise disjoint within each class. For example, individual of the *Device* cannot be instance of *Heater* and *Light* class at the same time. Subclasses of *State* class are pairwise disjoint, but their subclasses are also pairwise disjoint. For example, if we have *Device Light*, it can be a member of either *OnState* or *OffState*. Individuals of *Parameter* (let us say for example $CO_2$) are according to their current value member of *ParameterProperty* class. They can be *Good*, *Bad* or *Medium*. If they are not *Good*, they are also member of class *Deviation*, where we get information if they are too high or low, therefore are in class *TooHigh* or *TooLow*. Subclasses in *Deviation* and *Quality* are also pairwise disjoint.

## 4.2. Object-properties and data-properties

Properties are binary relations, whereas we have two type of properties. We can also define mathematical properties for ontology properties, such as transitive, symmetric asymmetric, reflexive, irreflexive, functional, inverse-functional, inverse to some property. We can also define sub-properties, which is similar to subclasses, meaning individuals linked with sub-property are also linked with super-property, inheriting all of super-property properties. Properties can also be disjoint, meaning that individual cannot be linked by disjoint properties at same time.

OWL defines two types of properties, object-properties and data-properties. Object-properties define relations between two classes or individuals. We can also define domain of the property and range of the property, meaning that relation must have individual of class defined in the domain linked to the individual that is a member of the class defined in range. For example, we define relation *influences* with domain of class *Action* and range of class *Parameter*. Instances of class *Action* may influence individuals of class *Parameter*. Full hierarchy of object-properties can be seen in Figure 4.2

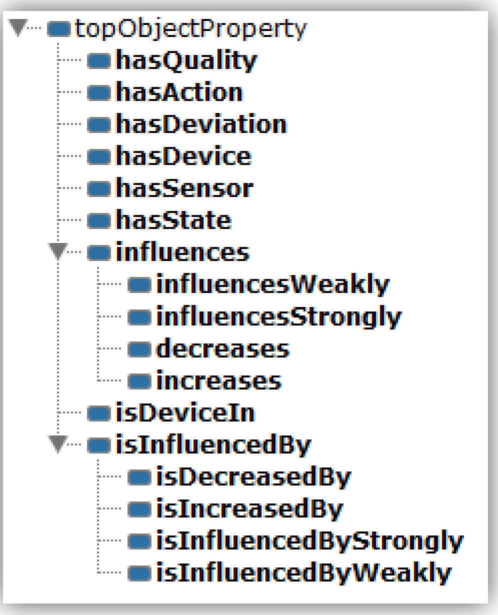
```
topObjectProperty
    hasQuality
    hasAction
    hasDeviation
    hasDevice
    hasSensor
    hasState
    influences
        influencesWeakly
        influencesStrongly
        decreases
        increases
    isDeviceIn
    isInfluencedBy
        isDecreasedBy
        isIncreasedBy
        isInfluencedByStrongly
        isInfluencedByWeakly
```

Figure 4.2 Hierarchy of object-properties defined in ontology.

Properties *hasAction* and *hasState* both have domain *Device*, but have range *Action* and *State* respectively. Properties *hasDeviation*, *hasQuality* have domain *Parameter* and range *Deviation* and *Quality* respectively. Relation *hasDevice* connects *Building* with *Device*. Property *influence* has domain *Action* and range *Parameter*. Same domain and range by definition apply to all sub-properties of property *influence*. Note that we defined some inverse relations: *influences* is inverse to *isInfluencedBy*, *hasDevice* is inverse to *isDeviceIn*. Relation *X* that is inverse to relation *Y*, has opposite domain and ranges. Property *influence* has sub-properties *increases* and *decreases*, which are disjoint, meaning that individual of *Action* cannot increase and decrease individual of class *Parameter* at the same time. Sub-properties *influencesWeakly* and *influencesStrongly* are also disjoint.

We also defined data properties, which connect individuals of classes to an actual chunk of data presented in Figure 4.3. The data property *hasValue* is a link between individual of *Parameter* (e.g. *Temperature*) and an actual value (e.g. 22). The data property *hasStateValue* is used for storing "raw" state value of individuals from class *Device* (e.g. *Heater*). Values are retrieved from the real sensor. For the purpose of development of Fit4Work ambient conditions monitoring, we implemented a smartphone application to enable the user to label the current status of device. For the final product, the state values will be provided from virtual sensors, which are machine-learning models trained to estimate the state of the devices.

Parameter values are measured with NetAtmo station (temperature, humidity, $CO_2$, external temperature, external humidity, noise) and one with smart phone (luminosity). According to state values of devices and values of parameters, reasoner infers in which *State* class the *Device* is in and in which *ParameterProperty* class *Parameter* is in according to SWRL rules covered in Section 4.3.

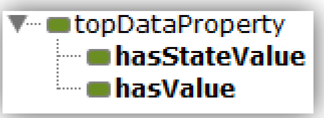

Figure 4.3 Hierarchy of data-properties defined in ontology

## 4.3. SWRL rules

Complex rules are expressed with SWRL (Semantic Web Rule Language) rules. They enable to insert so-called built-ins, where we can perform simple math equations, which is especially suitable with making comparisons between internal and external parameters (important when deciding how window action is influencing parameter temperature and humidity). It is important if external parameters are higher/lower than internal, because opening window cause air mixing and internal parameters are converging towards external. SWRL rule is built from body (antecedent part) and head (consequent part). Single unit of the SWRL rule is called atom and can be either true or false. Atom can be unitary (whether individual is part of a class), binary (whether relation holds true between two individuals) and in some cases also n-nary (in various built-ins). If all atoms in body part are true, it follows that also all atoms in head part are true. More about SWRL rules can be seen in [3].

### 4.3.1. State rules

With SWRL rules we implemented some simple state rules for devices, which depends on state values of devices. We defined class *State*, which has few subclasses to define states for various devices. *Device* individuals are then put in one of the subclasses of class *State*.

Some of devices has binary state values, on and off. In our setting, we have light and humidifier as those kind of devices. We construct rule, that individual of any of those classes is in class *OnState* if state value is 1 or *OffState* if state value is zero.

We also defined devices with 3 possible states: *MinState*, *MediumState* and *MaxState*. Those devices are heater and ventilation. We defined possible state values for devices ranging from 0 to 100, where if device has state value of 0, it belongs to *MinState* class, if it has value in open interval of 0 and 100, it is in *MediumState* class or it belongs to *MaxState* class, if it has state value of 100.

Air-condition device has state values ranging from 17 to 27. This set-points were chosen according to common sense on what are the minimum and maximal possible set-points for air-condition device. We defined 4 possible states to present air-condition device: It can be in *OffState*, if it is not turned on, *MinState*, if it has state value 17, *MaxState* if state value is of 27, and every other state value between 17 and 27 puts device in class *MediumState*.

```
hasStateValue(Heater, ?it), greaterThan(?it, -2.0f), lessThanOrEqual(?it, 0f)        -> Min(Heater)

hasStateValue(Heater, ?it), greaterThan(?it, 0.0f), lessThanOrEqual(?it, 99.0f)   -> Medium(Heater)

hasStateValue(Heater, ?it), greaterThan(?it, 99f), lessThanOrEqual(?it, 100.0f)  -> Max(Heater)
```

**Figure 4.4 Example of establishing state rules of Heater device**

First atom in every rule in Figure 4.4 defines variable that belongs to individual *Heater*. With second and third atom we created interval that state value can be in. Note that we set "dummy" bounds for *MinState* and *MaxState*. State values of device Heater can range from 0 to 100, so we can afford liberty of setting dummy bounds in order to achieve desired goal. The head (consequence) of every rule tells us, that individual will be member of class *MinState*, *MediumState* or *MaxState*. Classes are disjoint, so no individual can be member of more than 1 of the classes at all time. It can also be seen, regarding defined range, that individual heater will be member of exactly one class. Similar rules are defined on other devices. We also used 3-state protocol for device *Window*, since it can be closed, opened or half-opened. So individual of *Window* is in *MinState* if *Window* is closed, in *MediumState* if *Window* is half opened and in *MaxState* if *Window* is opened. We implemented initialization of rules in Java with OWL-API.

### 4.3.2. Action rules

We used rules to define which action *Device* individual has according to their *State*. If device is in one of the possible states, it has actions to go to other possible states. For example, if *Window* is in *MediumState*, it has possible actions *WindowActionOpen* and *WindowActionClose*. We wrote this rules in ontology directly with Protégé SWRL rule editor, example of one such rule can be seen on Figure 4.5.

```
HeaterMin(?heater)       ->  hasAction(?heater, HeaterIncrease)
HeaterMedium(?heater) -> hasAction(?heater, HeaterIncrease), hasAction(?heater, HeaterDecrease)
HeaterMax (?heater)      -> hasAction(?heater, HeaterDecrease)
```

**Figure 4.5 Example of one predefined SWRL rule, that defines which actions are possible from current state of Device individual. This example shows us Heater individual.**

### 4.3.3. Property of parameters rules

For assessment of the parameters, we created class *ParameterProperty* with two subclasses: *Quality* and *Deviation*. Parameter can be of bad, medium or good quality. If it is not of good quality, then it can be too high or too low (which are subclasses of class Deviation). We also established simple rules, to assign individual of class *Parameter* to proper subclasses of class *ParameterProperty*. Example of such rule can be seen on Figure 4.6. Similar rules are applied for other parameters.

```
hasValue(Humidity, ? x), greaterThan(?x, 60.0f)  -> TooHigh(Humidity)
hasValue(Humidity, ?x), greaterThan(?x, 70.0f), lessThanOrEqual(?x, 100.0f) -> Bad(Humidity)
```

**Figure 4.6 Example of SWRL rules, where we define range that Humidity is too high and it is in bad range**

Bounds for deciding, whether individuals of *Parameter* are in class *Good*, *Medium* or *Bad* are defined after expert recommendations (Figure 4.9). To initialize those rules, we implemented generic method *parameterRule* in Java with OWL-API. With this method, we are able to modify bounds, which may be usable, if we would like to adjust bounds according to user feedback. Writing from practical experience, when we were developing and testing system in winter time, many occupants were complying about temperature being too low, when it was in Good interval (on Figure 4.9 we can see that 21°C is still Good temperature, while it was too cold for some of the occupants in the office). On Figure 4.7 and Figure 4.8 we can see snippet from the code.

```
private void parameterRule(OWLOntology o,float lowerBound, float
upperBound, String qualityString, String parameterString)
```

**Figure 4.7 . We implemented method parameterRule, which takes OWL ontology o, and defines to which Quality will individual of Parameter belong to, given its current value.**

```
float lowerBound = 21f;
float upperBound = 23f;
float mediumBound = 2f;
parameterRule(o, -300, lowerBound-mediumBound, ":Bad",
":Temperature");
parameterRule(o, lowerBound-mediumBound, lowerBound, ":Medium",
":Temperature");
parameterRule(o, lowerBound, upperBound, ":Good", ":Temperature");
parameterRule(o, upperBound, upperBound+mediumBound, ":Medium",
":Temperature");
parameterRule(o, upperBound+mediumBound, 300, ":Bad",
":Temperature");
```

**Figure 4.8 Example from Java code, where we initialize SWRL rule in this case for Temperature**

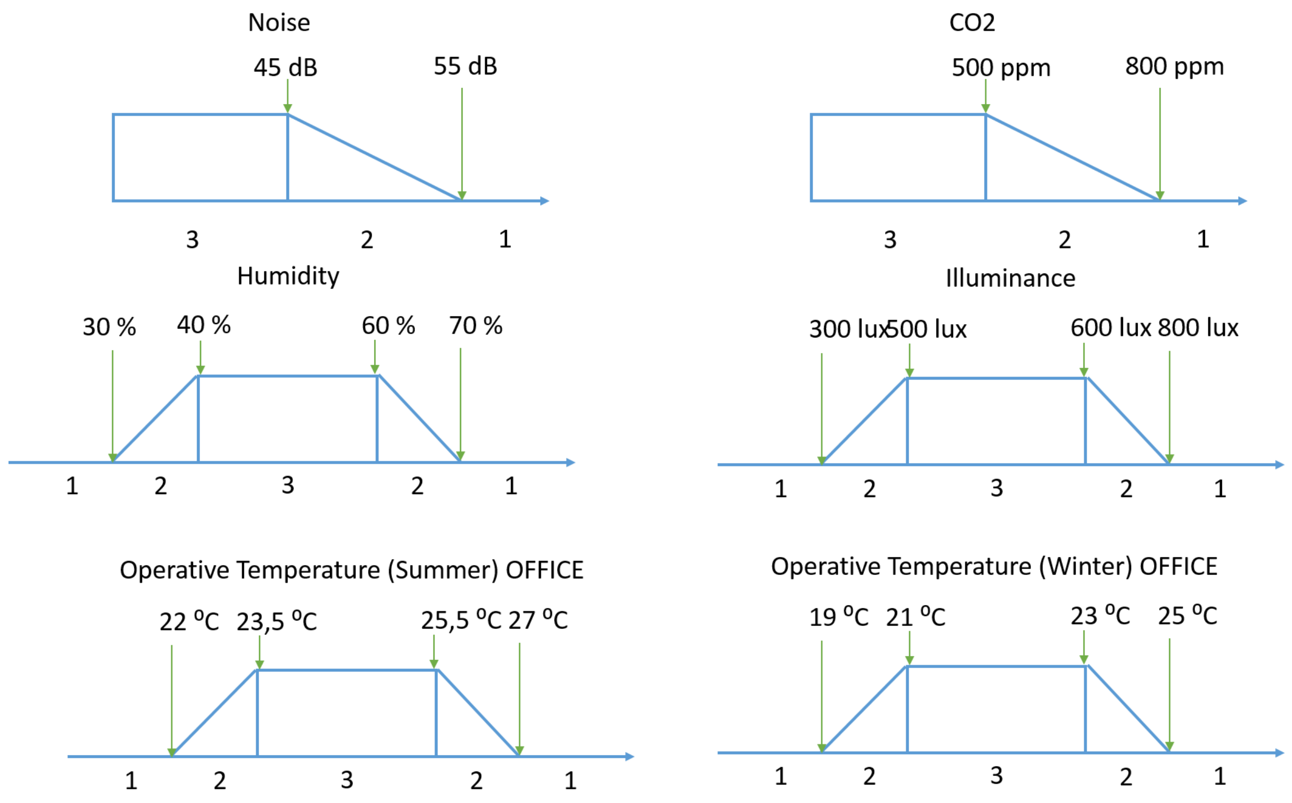

**Figure 4.9 Figure is showing bounds of the parameters Noise, CO2, Humidity, Illuminance and Temperature in winter and summer time. Intervals marked as number 3 are presenting zones, where single parameter have Good value. Sections marked with 2 are representing Medium zone, whereas sections with mark 1 present Bad zones.**

### 4.3.4. Influence rules (increase/decrease and influence strongly/weakly)

According to possible actions of devices, we have to define how actions affect certain parameters. We define in ontology itself rather simple rules, such as *HumidifierActionOn* increases *Humidity*, *HeaterActionIncrease* increases *Temperature*, etc. We used SWRL rules to define how window actions are influencing individuals *Temperature*, *Humidity* and *CO2*.

*hasValue(Temperature, ?it) , hasValue(ExternalTemperature, ?et), lessThanOrEqual(?et,?it)*

->

*decreases(WindowActionOpen, Temperature), increases(WindowActionClose, Temperature), decreases(WindowActionHalfOpen, Temperature)*

**Figure 4.10 Example of SWRL rule. It compares internal and external temperature and according to result, it determines wether actions with windows increases or decreases internal temperature.**

Figure 4.10 presents a rule with which we compare internal and external temperature and then establish relation *influence* between window actions and parameters. For individual *Humidity*, we have to derive additional individual *AbsoluteHumidity*, because NetAtmo station provide us only with relative humidity inside and outside. With additional information of temperature, which we also possess inside and outside value of, we are able to derive absolute humidity, which is then presented to ontology (we implement this in Java code). Similar rules as one described above were applied to create relation of influence between actions of window and *Humidity* parameter. All rules described above are written in SWRL rule-editor using Protégé.

We also define sub-property of *influence*, which is if action influence strongly or weakly given parameter. With other devices, except window, relations are straight-forward: *HeaterIncrease* influences *Temperature* strongly, *HeaterDecrease* influences *Temperature* weakly. *HumidifierOn* influences *Humidity* strongly, when *HumidifierOff* influences weakly. All air-condition actions influence *Temperature* strongly. We predefined described rules in SWRL rule editor.

Strength of influence of window action depends on external parameters. We used built-ins to derive absolute difference between external and internal parameters for *Temperature* and *Humidity* (note that we used absolute humidity). We implemented generic method for initialization of these rules with OWL-API, since we have to set bound, that determines how much has to be the difference between internal and external parameter to determine strength of influence.

```
hasValue(Temperature, ?it) , hasValue(ExternalTemperature, ?et), subtract(?diff, ?it,?et), abs(?adiff,
?diff), greaterThan(?adiff, 5f) -> influencesStrongly(WindowActionOpen, Temperature)
```

**Figure 4.11 Example of SWRL rule how strongly Window influences Temperature**

In rule on Figure 4.11 we express that if difference between internal and external Temperature is more than 5°C, *WindowActionOpen* influence strongly on *Temperature*. If it is smaller, than similar rule apply, except this time *WindowActionOpen* influences weakly on *Temperature*. Similar rules are applied for *WindowActionClose* and in analog manner rules for *Humidity* apply.

## 4.4. Putting all pieces together and querying with SPARQL-DL

Figure 4.12 presents the whole layout of the ontology. At initialization of the system, we create new individual *Room* of class *Building* and list which devices it possesses. We also tell ontology which state values devices has and values of *Parameters*. With OWL-API, we initialize some SWRL rules, some of them are already predefined within ontology. Reasoner then, using rules and defined axioms of ontology, infers and puts ontology in a valid state. We want to know from ontology, which actions can be taken in order to improve parameters, which are in medium or bad zone. So we find parameter, that is in not good zone, check if it is too high or too low and appropriately find actions that influence that parameter in right way (if it is too high, we must find actions that decreases that parameter and vice-versa). Action that we get from this special query are then presented as output of the ontology. This special query is called SPARQL-DL query [5, 6, 7], whose predecessor (SPARQL) was firstly designed to answers RDF queries [4]. We can see example of SPARQL-DL query, used in the application in Figure 4.13. In example we search for individuals of class *Action* that *influences* individuals of class *Parameter*, *Bad* and *TooLow*. We use similar queries to consider parameters that are too low and in medium zone, and parameters that are too high and are in medium and bad zone, which results in 4 queries. All found actions are then presented to the next step of the system.

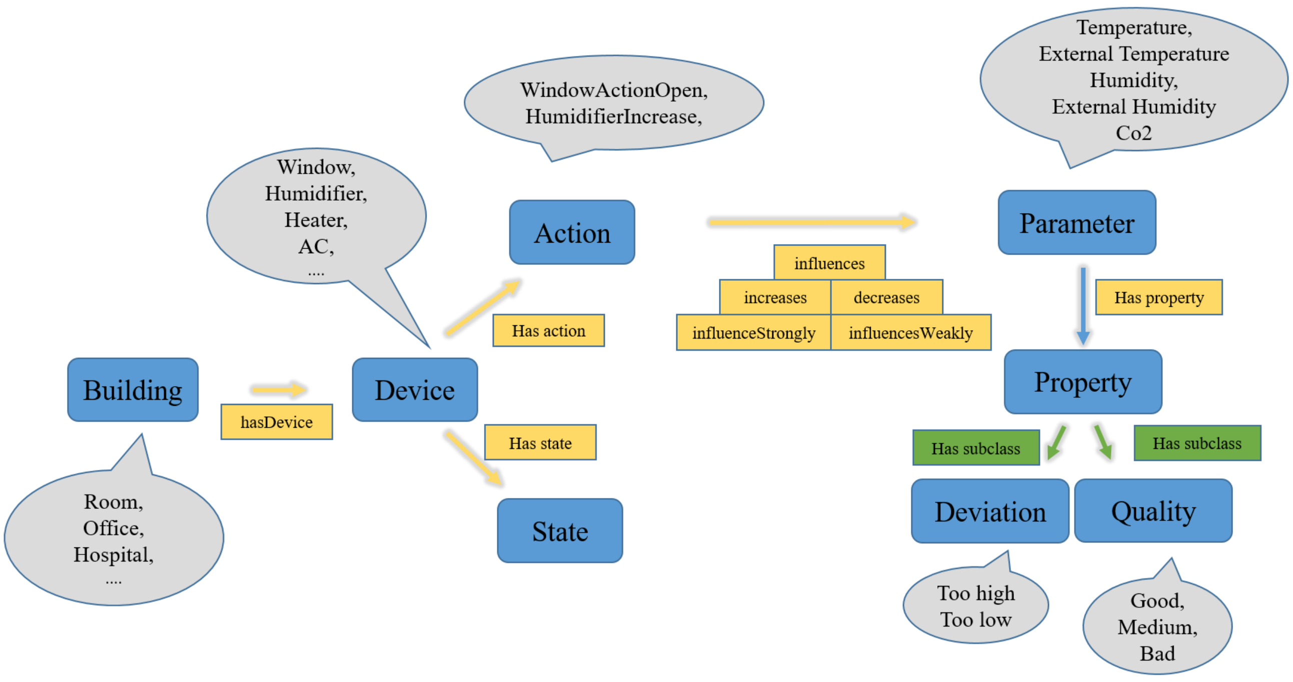

Figure 4.12 Overview of the ontology

```
PREFIX onto: <http://www.semanticweb.org/anton/ontologies/2015/8/fit4work-ontology#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?par ?action ?device ?influenceType
        WHERE{
                ?par a onto:Bad ;
                        a onto:TooLow ;
                        onto:isInfluencedBy ?action ;
                        ?influenceType ?action.
                ?influenceType rdfs:subPropertyOf onto:isIncreasedBy .
                ?device onto:hasAction ?action .
                onto:Room onto:hasDevice ?device
                }
```

Figure 4.13 Example of SPARQL-DL query in application

## 5. Discussion

In this deliverable we present the knowledge base used for recommendations presented in deliverable 3.3.1/3.3.2. The knowledge is either in form of list of most relevant activities in elderly, daily and weekly requirements, in form of a set of exercises in case of functional fitness exercises and stress relief exercise and in case of ambient conditions in form of an ontology.

Future work includes updating the knowledge base for each module, with the use of tools discussed herewith and in relevant sections of deliverable 3.3.1/3.3.2.

## 6. Bibliography

1. OWL 2 and SWRL Tutorial. http://dior.ics.muni.cz/~makub/owl/ (Online 3.4.2016)
2. Web Ontology Language. https://en.wikipedia.org/wiki/Web_Ontology_Language (Online 3.4.2016)
3. SWRL Language FAQ. http://protege.cim3.net/cgi-bin/wiki.pl?SWRLLanguageFAQ (Online 3.4.2016)
4. SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/ (Online 3.4.2016)
5. I. Kollia, B. Glimm, I. Horrocks. Answering Queries Over OWL Ontologies with SPARQL. http://webont.org/owled/2011/presentations/4Kollia.pdf (Online 3.4.2016)
6. I. Kollia, B. Glimm, and I. Horrocks. 2011. SPARQL query answering over OWL ontologies. In Proceedings of the 8th ESWC on The semantic web: research and applications, 382-396.
7. Sirin, Evren, and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-DL. OWLED. Vol. 258. 2007.
8. Matthew Horridge, Sean Bechhofer. The OWL API: A Java API for Working with OWL 2 OntologiesOWLED 2009, 6th OWL Experienced and Directions Workshop, Chantilly, Virginia, October 2009
9. The OWL API, http://owlapi.sourceforge.net/ (Online 3.4.2016)
10. Knublauch H., Fergerson R. W., Noy N. F., Musen M. A. 2004. The Protégé OWL plugin: an open development environment for Semantic Web applications. In Third International Conference on the Semantic Web, Hiroshima, Japan, pp. 229--243.
11. E. Sirin, B. Parsia, B. C. Grau, A. K., Y. Katz. 2007. Pellet: A practical OWL-DL reasoner. Web Semant. 5, 2, 51-53. http://dx.doi.org/10.1016/j.websem.2007.03.004