hackAIR

OPEN PLATFORM

D4.2: Semantic integration and reasoning of environmental data

WP4 – Data fusion model and reasoning services

Document Information

Grant Agreement Number	688363	Acronym	hackAIR	
Full Title	Collective awareness platform for outdoor air pollution			
Start Date	1st January 2016	Duration	36 months	
Project URL	www.hackAIR.eu			
Deliverable	D4.2 – Semantic integration and reasoning of environmental data			
Work Package	WP4 – Data fusion model and reasoning services			
Date of Delivery	Contractual	30th June 2017	Actual	28th June 2017
Nature	Report	Dissemination Level	Public	
Lead Beneficiary	CERTH			
Responsible Author	Marina Riga (CERTH)			
Contributions from	Anastasia Moumtzidou (CERTH), Stefanos Vrochidis (CERTH), Ioannis Kompatsiaris (CERTH), Panagiota Syropoulou (DRAXIS)			

Document History

Version	Issue Date	Stage	Description	Contributor
0.1	31/03/2017	Draft	Template provided	Panagiota Syropoulou (DRAXIS)
	07/04/2017	Draft	Document Structure	Marina Riga, Stefanos Vrochidis (CERTH)
0.2	24/04/2017	Draft	Writing content	Marina Riga (CERTH)
0.3	02/06/2017	Draft	Sent for internal review (CERTH)	Marina Riga, Anastasia Moumtzidou, Stefanos Vrochidis (CERTH)
1.0	16/06/2017	Draft	Revised draft	Marina Riga (CERTH)
1.0	16/06/2017	Draft	Sent for internal review (DRAXIS)	Panagiota Syropoulou (DRAXIS)
2.0	26/06/2017	Draft	Review feedback, revised draft	Marina Riga (CERTH)
3.0	27/06/2017	Final	Final version	Marina Riga (CERTH)

Disclaimer

Copyright message

Table of Contents

Table of Figures

Table of Tables

Glossary

Abbreviation/Acronym	Meaning
AOD	Aerosol Optical Depth
API	Application Programming Interface
AQ	Air Quality
AQI	Air Quality Index
CQ	Competency Question
DS	Decision Support
DSS	Decision Support System
JSON	JavaScript Object Notation
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
KB	Knowledge Base
MOT	Mean Of Transport
ORSD	Ontology Requirements Specification Document
OWL	Web Ontology Language
PM	Particulate matter
$PM_{2.5}$	Particulate matter up to 2.5 micrometres in diameter
PM_{10}	Particulate matter up to 10 micrometres in diameter
PDL	Problem Description Language
RDF	Resource Description Framework
RESTful	Representational State Transfer
RS	Recommendation System
SME	Small and Medium-sized Enterprises
SPARQL	Simple Protocol and RDF Query Language
SPIN	SPARQL Inferencing Notation
SWRL	Semantic Web Rule Language
UI	User Interface

W3C	World Wide Web Consortium
XML	Extensible Markup Language

1 Executive summary

This document reports the research and implementation that has been conducted for the task of semantic integration and reasoning of environmental and user-profile data. More specifically, the deliverable covers the following:

- The creation of an *ontological framework* for the formalisation of heterogeneous data that are targeted to be stored in the hackAIR knowledge base (KB); the latter aims to serve as a storage module of the content and existing relations, representing information about the user profile and needs, as well as the existing environmental data of areas of interest (Section 3).
- The formulation of a *problem description language* that defines the structure and concepts for translating a request for recommendation from the user into a representation expressed in terms of ontology concepts and relations (Section 4).
- The development of a *service* that handles the dynamic population of the KB at run time (i.e. when a user submits the request for recommendation); the input data will be delivered by the hackAIR User profile module and the Data fusion module (Section 5).
- The implementation of a *reasoning mechanism* for providing personalised decision support services, with respect to specific characteristics of user profiles and to existing air quality (AQ) conditions (Section 6).

We investigate how involved data are generally represented as ontological concepts in existing domains of common interest with the project domains (environmental-, people-, health-, location-, recommendations- related) and we build our own integrated schema for the alignment of heterogeneous data in an ontological KB. Environmental and user-specific data are considered as heterogeneous due to their difference in nature (textual and numerical) and source (i.e. derived from different modules).

On the basis of the ontological framework, we specify which entities can be utilised in the process of transforming the unstructured data content of a user request for recommendation into an ontology-based representation for further utilisation by the reasoning module. The mapping between unstructured and structured content is made automatically, with the use of an implemented web-service, without interfering the user in such a complex task that requires specific technological expertise.

We discuss different approaches for performing reasoning on top of the newly populated data in the hackAIR ontology and we outstand why a SPIN[1] rule-based reasoning implementation is more efficient for the inference of new knowledge, by taking advantage of the data and semantics represented in the ontological framework. For the need of inferring user-profile driven recommendations, we specify which characteristics of the user profiles are considered as more important in the recommendation process, as well as what kind of suggestions and tips will be provided to the users, with respect also to existing environmental issues.

[1] http://spinrdf.org/

2 Introduction

One of the main targets of hackAIR project is to develop a methodology for synthesising heterogeneous air quality data collected in order to generate meaningful information, personalised to the requirements of citizens. This objective involves: a) the development of a data fusion algorithm for merging air quality information from various sources to provide value-added map products of air quality to citizens and to offer air quality-aware personalised services to the public, b) the semantic integration of environmental and user-specific data, and c) the reasoning and decision support strategies involved data for the generation of personalised recommendations. The first part is targeted in Deliverable 4.1 "*Developed and tested data fusion algorithm for use in the pilot study activities WP7*", while the remaining define the main objectives of this deliverable - D4.2 "*Semantic Integration and reasoning of environmental data*". Both the semantic representation and the reasoning framework of the current deliverable will be of significance to the final services of WP5 "*Development of the hackAIR platform*".

To achieve the objectives of this deliverable, it is necessary to represent in a common way the heterogeneous information captured by hackAIR including the user profile and needs (User profile Module) and environmental data (Data fusion Module) in order to provide personalized decision support services. This task involves a set of steps that should be considered and which are discussed in the current deliverable. Specifically, a) the design of an ontological framework, b) the formulation of a problem description language (PDL), c) the dynamic population at run time of the knowledge base (KB) and d) the development of reasoning techniques. The main workflow that involves processes related to the objectives of the current deliverable is presented in Figure 1.

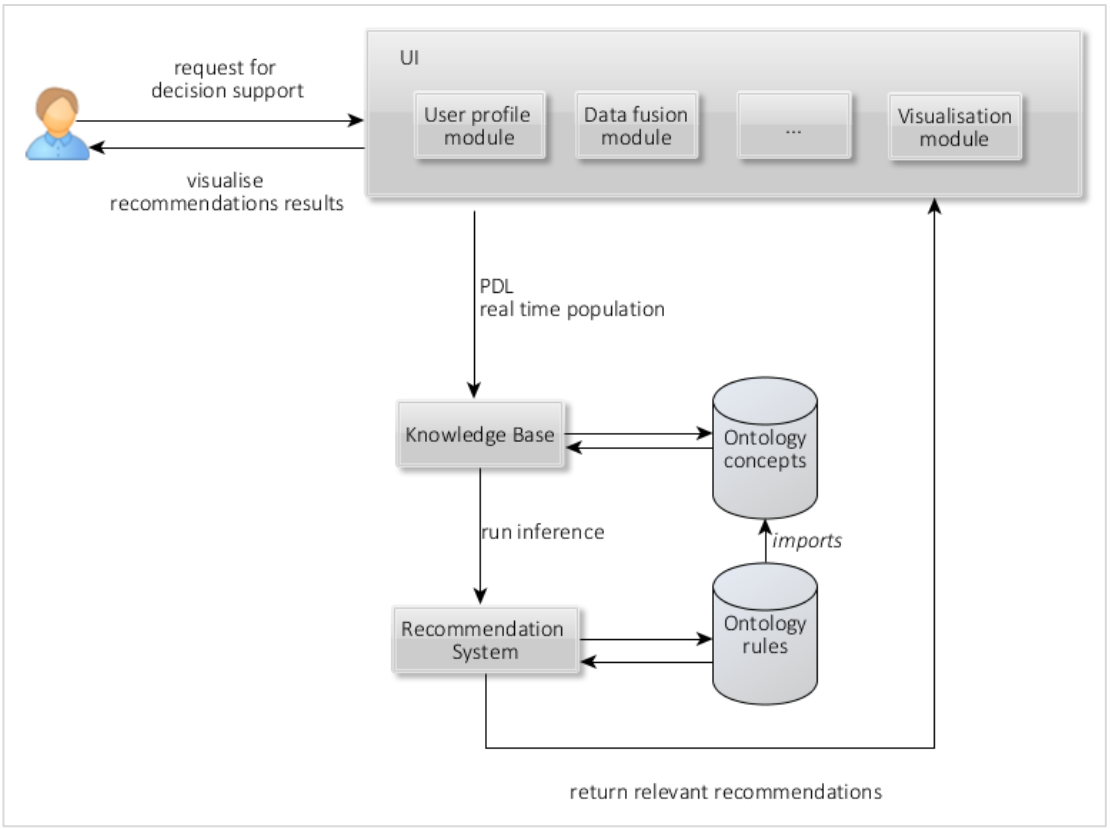


Figure 1 – User triggered processing pipeline

More specifically, in Figure 1 we present all involved hackAIR modules, as well as their between communication, for performing semantic integration of user-profile and environmental data and reasoning processes for providing personalised recommendations to the user. The main workflow involves the following steps: (1) the requests for decision support through the use of the hackAIR platform; it is considered that the user has specified his/her

preferences and characteristics beforehand in a different process (create user profile), which is out of the scope of the current task; (2) all significant data for decision support (i.e. user profile and fused AQ data) are gathered at run-time (when the user submits the request) from corresponding hackAIR modules. These data are aimed to be populated in the hackAIR KB for further manipulation; (3) through the adoption of PDL guidelines by relevant hackAIR services, previously collected data are transformed into relevant ontology notions; (4) then, the recommendation system (RS) is triggered and rules stored on top of the ontology perform the inference at run-time; (5) inferred recommendation(s) are returned to the user through the visualisation module of the hackAIR UI.

In order to cover the aforementioned functionality, specific research and development has been conducted and presented in the current deliverable. Initially, we present the most established methodologies used for designing and implementing ontological frameworks and based on their characteristics as well hackAIR needs and we select the method that covers our needs. Then, we present the scope, intended uses and requirements of the hackAIR ontology. Based on the outcome of the analysis we define the entities and concepts that the ontology covers, together with standards and parts of existing ontologies that can be aligned to the hackAIR concepts. Finally, we present some ontology metrics and results to evaluate the structure, quality and consistency of the developed ontology.

In the sequel, we present the hackAIR Problem Description Language (PDL), giving in detail the structure and details on how the request of the user is formulated so as to be properly submitted for decision support by the hackAIR RS. We describe the requirements that were taken into account in the design process of the PDL, the main entities of the PDL and we conclude with example PDL representations in real use case.

Then, we discuss the process of manipulating a user request for decision support. In order for the hackAIR RS to efficiently operate the reasoning process, the user-profile data (user characteristics, details and needs, provided by the User profile Module) and environmental data (provided by the Data fusion Module) should be populated in the ontology-based KB upon request. For this reason, we develop a web-service that performs real-time conversion of data into relevant ontology representations and dynamic population (i.e. when the user submits a request) of the KB, by efficiently adopting the hackAIR PDL and ontology notions.

Next, we present the reasoning requirements and framework. We specify the nature and context of recommendations provided for decision support; two types of recommendations are supported: (i) *tips of the day*, i.e. general daily advice on how to improve the ambient air quality (AQ) and to reduce individuals' air pollution production, and (ii) *personalised recommendations*, i.e. user profile-driven advice with respect to its characteristics (age, health sensitivity, preferred activity (-ies)) and to existing AQ conditions (expressed in 4 different quality states), on how to protect from air pollution.

Since the recommendation process takes into account the user profile needs as well as the environmental data, the reasoning framework has to operate on top of the semantic relations (ontology schema) and individuals (populated data) that are represented in the KB. So, in order for the hackAIR RS to comply with the nature of the hackAIR KB, ontology-rules are developed with an extensive use of SPIN (SPARQL Inferencing Notation) standard.

The document is structured as follows: Section 3 presents the ontological framework developed for hackAIR; Section 4 discusses the problem description language, its functionality and characteristics; Section 5 discusses the dynamic population of heterogeneous data (user profile characteristics, environmental measurements) at the time that the user submits a request for decision support; Section 6 presents the reasoning techniques, with details on the architecture, methodology and implementation followed. Finally, Section 7 concludes this deliverable with a summary of results and future work.

3 hackAIR ontological framework

To support the understanding, sharing and reuse of knowledge (information) among systems, it is useful to define a common vocabulary in which shared knowledge is represented in a formal way. The specification of a representational vocabulary for a shared domain of discourse is called *ontology* [Gruber, 1993]. The ontology enables the modelling of knowledge by defining a set of representational primitives, which typically are: ***classes*** (objects, concepts and other entities) that are assumed to exist in some area (domain) of interest, and their ***properties*** (attributes, i.e. relationships that hold among them). Their expressivity and level of formalisation depend on the knowledge representation language used.

Within the Semantic Web, i.e. extension of the current Web that aims to augment web resources by attaching to them metadata that describe meaning in a formal, machine-understandable way, ontologies play a key role. In this effort, the Web Ontology Language (OWL) has emerged as the official W3C recommendation for creating and sharing ontologies on the web [Bechhofer, 2009]. OWL is the most well-established ontology language, since it facilitates greater machine interpretability of represented content than that supported by XML, RDF, and RDF Schema (RDF-S), by providing additional vocabulary, along with formal semantics [W3C, 2004].

This chapter presents the scope, intended uses and requirements of the hackAIR ontology. We then describe in detail the structure, the entities and concepts that the ontology covers, together with standards and parts of existing ontologies that the hackAIR ontology can potentially adopt. We demonstrate how unstructured data are described according to defined hackAIR ontology notions, and finally, we present some well-known ontology metrics and results to evaluate the structure, quality and consistency of the developed ontology.

3.1 Ontology engineering process

For the design and implementation of the hackAIR ontological framework we studied the most established methodologies and approaches, including *Sensus* [Swartout et al., 1997], *KACTUS* [Bernaras et al., 1996], *DOGMA* [Jarrar and Meersman, 2008], *METHONTOLOGY* [Fernandez et al., 1997], *DILIGENT* [Pinto et al., 2004], *On-To-Knowledge* [Sure et al., 2004], *Cyc* [Lenat and Guha, 1989], *Unified* [Uschold, 1996], *Grüninger and Fox* [Grüninger and Fox, 1995], and others. While all examined methodologies presented interesting perspectives towards building ontologies either from scratch or by additionally inheriting existing ones, we decided to select and apply the **NeOn ontology development methodology** [Suárez-Figueroa et al., 2009]. NeOn is a scenario-based methodology that covers various cases of ontology development. In detail, the methodology (i) guides the ontology engineer to define efficiently the requirements and characteristics of the ontology, (ii) takes into account the existence of multiple ontologies in ontology networks, and (iii) supports the reuse/reengineering of knowledge resources. It consists of the following components:

- **The NeOn Glossary** - a well-established glossary that includes 59 predefined processes and activities. Its purpose is to provide a standard vocabulary, created by ontology experts that can be used for well-described and structured processes.
- **Scenarios for building ontologies and ontology networks** - unlike other methodologies, NeOn approaches a variety of scenarios for ontology engineering, while each scenario is decomposed into different processes or activities.
- **Two ontology network life cycle models** - these models, named the *Waterfall Model* and the *Iterative/Incremental Model* indicate how to establish the development processes and activities.
- **A set of methodological guidelines for processes and activities** - These are specific guidelines in order to fulfil the activities and processes mentioned in the NeOn Glossary.

The selection of NeOn was based on a thorough comparison among methodologies, using a diverse set of criteria. These criteria, along with the comparison results (Table 1) are presented below, in order to document our decision:

- **Well-documented** - shows the depth and details provided for each process and guideline of the methodology;
- **Reusability** - shows whether the reuse/reengineering of existing ontologies is supported by the methodology;
- **Dynamic /Adaptive** - defines the level of adaptability to various cases of development;
- **Structured representation of requirements** - shows whether the methodology incorporates a structure that formally describes the development requirements.

Table 1 – Comparison of existing Ontology Engineering Methods

Methodology	Well-documented	Reusability	Dynamic/ Adaptive	Structured representation of requirements
Sensus	Medium	Yes	Low	No
KACTUS	Low	Yes	Low	No
DOGMA	High	No	Low	Tuples[2]
METHONTOLOGY	Medium	Yes	Low	No
DILIGENT	Medium	No	Low	No
On-To-Knowledge	High	No	Low	No
Cyc	Medium	Yes	Low	No
Unified	Low	No	Low	No
Grüninger and Fox	High	No	Medium	CQs[3]
NeOn	**High**	**Yes**	**High**	**ORSD[4]**

3.1.1 Ontology Requirements Specification Document (ORSD)

To elicit the requirements that the hackAIR ontology should satisfy, we followed the guidelines proposed in [Suárez-Figueroa et al., 2009] for the NeOn methodology. A key process in the aforementioned method is to identify beforehand which requirements and specifications the ontology should fulfil. In other words, it leads the ontology expert to specify in systematic way information about:

[2] In DOGMA framework, a tuple is a description of conceptual relations in the form <γ: Term1, Role, InvRole, Term2>, where Term1 and Term2 are linguistic terms, γ is a context identifier, and Role/InvRole are lexicalisations of the paired roles in any binary relationship; for each pair (γ, Term) is assumed to refer to a uniquely identifiable concept [Jarrar and Meersman, 2008].
[3] CQs stands for the term *Competency Questions*, i.e. a list of questions that the ontology should be *competent* to answer [Grüninger and Fox, 1995]. These CQs are just a sketch and there is no need to be exhaustive.
[4] ORSD is the abbreviation of the Ontology Requirements Specification Document, which is explained in detail in Section 3.1.1.

- the **purpose**, i.e. the main goal of the ontology;
- the **scope**, meaning the general coverage and the degree of detail in the ontology;
- the intended **end-users**;
- the potential **uses** of the ontology and its relevant features;
- the functional and non-functional **requirements**, attributed in detail, with the use of competency questions (CQs) and the analysis of their answers;
- a glossary of **terms**, significant for defining the ontology main entities to be represented.

This requirement specification methodology is documented in a template-based report called *Ontology Requirements Specification Document* (ORSD). For the needs of the project, and for developing the first version of the hackAIR ontology, we developed the ORSD presented in Table 2. The requirements of the hackAIR ontology were specified through relevant consortium discussions in cooperation with the user partners and are also aligned to user and technical requirements analysis presented in D2.2 [hackAIR D2.2, 2016].

Table 2 – The ORSD for the hackAIR ontology

hackAIR Ontology Requirements Specification Document	version 1.0
1	Purpose
	The overall goal of the hackAIR ontology is to provide an operational framework for the orchestration of heterogeneous (environmental-, health-, user profile- related) data in order to support user-oriented decision support services. The hackAIR platform aims to integrate an ontology-based reasoning module for environmental information and recommendations delivery, with respect to: (i) personal health and user preferences (activities, daily routine, etc.), and (ii) current AQ conditions of the location of interest. The ontology will provide the vocabulary and semantics that represent and generate all information needed in order for the hackAIR system to afford its end-users' services.
2	Scope
	The hackAIR ontology will be the basic framework of the hackAIR knowledge base (KB), for storing user profile data coming from the User Profile Module and air quality observations coming from the Data Fusion Module. The decision support (DS) module will directly communicate with the KB in order to gather relevant information and to infer proper recommendations to the users. Based on these facts, the ontology should formally represent: • environmental content, and especially air quality related data, such as air pollutant measurements, air quality index values, etc.; • user profile details, including age, gender, health status or any health sensitivity that can be aggravated by severe air quality conditions; • relationship data, i.e. connections between different user profiles; • geographical location data; • user interested activities, meaning daily or systematic or preferred outdoor activities;

	• description of the user's request for decision support, i.e. query related to a preferred activity, or related to his/her exposure to existing air quality conditions, etc.; • general warnings, tips or more personalised recommendations to be reported to the users; • knowledge of environmental and health experts, in terms of existing environmental conditions and their impact on public health. The need is to efficiently encode their expertise, in the form of rules and recommendations, through the relevant ontology concepts.
3	Implementation Language
	The ontology should be implemented in OWL 2, the officially recommended language for knowledge representation in the Semantic Web.
4	Intended End-Users
	The main end-user categories as well as their main characteristics are briefly reported below: • User 1. *General public*, individuals interested in receiving information about existing environmental conditions, and in supporting their actions/decisions during severe environmental conditions that can be potentially harmful to their personal health or the health of third parties; • User 2. *People with health sensitivities* and *people who perform outdoor activities*, who are particularly interested in receiving up-to-date air quality measurements for their place of interest, and also getting feedback in the form of guidance (advice messages) for limiting their exposure to hazardous conditions; • User 3. *Public administrators,* policy/decision makers, interested in receiving information on air quality for professional reasons, i.e. public health, legal issues, protection or proactive purposes, environmental awareness raising, environmental activists; • User 4. *Environmental experts*, interested in: (i) being informed with respect to the evolving air quality conditions, (ii) notifying interested groups when needed, (iii) monitoring short- and long- term effects for estimating potential health risks, and (iv) defining accurate action plans according to actual air quality observations on the location of interest; • User 5. *Communities of citizens and activists organisations*, interested in being timely and accurately informed about AQ conditions, and to motivate their members. • User 5. *Data contributors*, interested in utilizing hackAIR services so as to provide relevant air quality information (measurements from sensors, images, social media data, etc.); • User 6. *Technological experts*, researchers, individual app developers or SMEs, interested in adopting the ontology model and to expand relevant decision support systems.

	It should be noted that the user groups considered in the hackAIR system, i.e. end-users (general public, health sensitive persons, data contributors) and stakeholders (public administrators, environmental experts) will not utilise directly the hackAIR ontology; their position is to take advantage of ontology utilities through relevant services (AQ information, recommendations, etc.) of the hackAIR platform. On the other hand, technological experts could be in position to use directly the hackAIR ontology, according to their needs.
5	Intended Uses
	Since the end-users will not necessarily interact with the hackAIR ontology, in a direct way, the intended uses are mainly application or service oriented. We identify the following ones: • Use 1. To store and retrieve the details of the profile of the users. • Use 2. To store and retrieve the detailed measurements of air quality related data, for the monitored locations. • Use 3. To store the details of the DS problem (request for recommendation) that is submitted to the recommendation system (RS). • Use 4. To map existing measurements into established air quality standards (AQI, AOD data, etc.). • Use 5. To serve as the conceptual model for the representation of the significant characteristics of individuals that may group them as sensitive health persons. • Use 6. To retrieve, adopt or extend the sensitive health groups that are supported by the RS, as well as their relation to profile-based recommendations. • Use 7. To provide a shared vocabulary for the representation, communication and exchange of information related to the resources of (i) air quality and observations, (ii) users' profiles and interested activities, and (iii) recommendations for behavioural change and for limiting people's exposure to air pollution.
6	Ontology requirements
	a. Non-Functional Requirements
	• NFR1. The ontology needs to be based on well-established technologies and ontology engineering methodologies. • NFR2. The ontology should reuse existing ontologies and available standards, whenever possible. • NFR3. The implemented ontology should be well documented, adaptable and extensible for future use. • NFR4. The ontology should support representation of its content in the English language.
	b. Functional Requirements: Groups of Competency Questions (61 CQs)

	CQs Group 1: Environmental data
	• CQ1. What are the main categories of environmental data? Air pollutants and related AQI values. • CQ2. What types of air pollutants are considered in the hackAIR system? Only measurements of particulate matter ($PM_{2.5}$ and PM_{10}). • CQ3. *What is the nature of the environmental data considered?* Numerical data (measurements), images, text, fused data. • CQ4. *What is the source of the environmental data considered?* Sensors, open services from existing air quality stations, social media, sky-depicting images uploaded to the hackAIR mobile app. • CQ5. *What is the type of environmental node?* Stations, web-sites and social media platforms, web-services and applications, sensors. • CQ6. *What is the type of area where the environmental node is located?* It could be almost anywhere: rural, suburban, urban, sea regions, high-traffic areas, industrial, etc. • CQ7. *Which type of air quality data are provided to the hackAIR KB and RS?* Single values for air pollutant (PM) concentrations or relevant aerosol optical depth (AOD) values, derived from the Data Fusion Module. • CQ8. What is the unit measurement for numerical values of considered air pollutants? Micrograms (one-millionth of a gram) per cubic meter ($\mu g/m^3$). • CQ9. How many qualitative scales (indices) are considered for describing air quality? Four (4). • CQ10. Which are the qualitative values of air quality indices? Very good, good, medium, bad. • CQ11. To what numerical value range do **very good** $PM_{2.5}/PM_{10}$ index correspond? 0-10 $\mu g/m^3$ and 0-20 $\mu g/m^3$ correspondingly. • CQ12. To what numerical value range do **good** $PM_{2.5}/PM_{10}$ index correspond? 11-25 $\mu g/m^3$ and 21-50 $\mu g/m^3$ correspondingly. • CQ13. To what numerical value range do **medium** $PM_{2.5}/PM_{10}$ index correspond? 26-35 $\mu g/m^3$ and 51-70 $\mu g/m^3$ correspondingly. • CQ14. To what numerical value range do **bad** $PM_{2.5}/PM_{10}$ index correspond? Equal or above 36 $\mu g/m^3$ and 71 $\mu g/m^3$ correspondingly. • CQ15. To what numerical value range do **very good** AOD PM index correspond? Above 0 and less or equal to 0.14 value. • CQ16. To what numerical value range do **good** AOD PM index correspond? Above 0.14 and less or equal to 0.34 value. • CQ17. To what numerical value range do **medium** AOD PM index correspond? Above 0.34 and less or equal to 0.44 value. • CQ18. To what numerical value range do **bad** AOD PM index correspond? Above 0.44 value.
	CQs Group 2: User profile data

- CQ19. What kind of information does the platform request from the users when creating a user profile? Demographic data, health related data, activity related data.
- CQ20. What kind of information is collected from users automatically? IP, location, language, mobile device info, …
- CQ21. *Which demographic data are collected?* Name, username, year of birth, gender, city/country of interest (user location).
- CQ22. *Which health related data are collected?* Health sensitivity (true/false), pregnant (true/false).
- CQ23. *Which activity related data are collected?* The user may select one or more preferred outdoor activities from a predefined list, which will be included in a request to the RS.
- CQ24. *Any other data requested for the user profile?* Email, which is mandatory for registration.
- CQ25. What is the gender of the user? Male, female or other.
- CQ26. Which are the general user categories that are considered for providing recommendations from the hackAIR system? Direct and indirect users.
- CQ27. *Which are considered as **direct users** in the hackAIR RS?* Individuals that create the profile and request for personalised recommendations for themselves.
- CQ28. *Which are considered as **indirect users** in the hackAIR RS?* Individual(s) for whom a direct user creates additional profile(s) and requests for additional recommendations according to their characteristics.
- CQ29. Which are the main categories a direct/indirect user may belong to, according to his/her year of birth? Infant, toddler, child, young, adult, middle-aged, elderly.
- CQ30. *Which are the main categories a direct/indirect user may belong to, according to his/her health sensitivity?* The user may or may not belong to sensitive health group. He/she will be considered as a sensitive-health person to the system.
- CQ31. Which are the main categories a direct/indirect user may belong to, according to his/her preferred activities? Sports general, Sports walking, Sports picnic, Outdoor working person.
- CQ32. *Will the user belong into only one user category?* No, the user may belong into one or more user categories. For example a user may be an adult, female, pregnant and outdoor working person.
- CQ33. To what numerical age do **infant** users correspond? Less or equal to 1 year old.
- CQ34. To what numerical age do **toddler** users correspond? Above 1 and less or equal to 3 years old.
- CQ35. *To what numerical age do **child** users correspond?* Above 3 and less or equal to 10 years old.
- CQ36. *To what numerical age do **young** users correspond?* Above 10 and less or equal to 17 years old.
- CQ37. *To what numerical age do **adult** users correspond?* Above 17 and less or equal to 40 years old.
- CQ38. To what numerical age do **middle-aged** users correspond? Above 40 and less or equal to 60 years old.
- CQ39. To what numerical age do **elderly** users correspond? Above 60 years old.

	• CQ40. *Which types of user could be considered as the target audience of the hackAIR recommendation services?* People with a health problem or with higher health risk related to air pollution, user groups with an interest in air quality and public health, parents with children, elderly people, pregnant women, elderly people, athletes or people exercising outdoors, people working outdoors, those who commute to work on a bike or walk, scientific communities, environmental activists, teachers and students, policy makers, etc.
	CQs Group 3: Location data
	• CQ41. At which granularity the location information is considered? It depends on the source. • CQ42. Does hackAIR system takes into account the exact location coordinates of the user when creating a user profile? No. • CQ43. *Does any other hackAIR service takes into account the coordinates of user's location?* Coordinates are taken into account only when collecting user-generated data (open hardware measurements or sky-depicting images). • CQ44. What type of information related to user's location is requested for creating the user profile? City name and country name.
	CQs Group 4: Activities data
	• CQ45. *Which are considered as physical outdoor activities?* Jogging, walking, biking, picnic, playing in the park, playing tennis, working, travelling, etc. • CQ46. *What is the travel modality the user can choose?* By car, by bicycle, by foot, by public means of transport, etc. • CQ47. *Which are the activities the user can request for decision support?* Making some physical outdoor activity (working outdoors, walking, having picnic, doing outdoor sports in general).
	CQs Group 5: Problem Description and Request data
	• CQ48. *How a request for Decision Support (DS) is applied to the hackAIR system?* The request is submitted by the user through a relevant interface of the hackAIR system. • CQ49. *What is the process for submitting a request to the DS system?* The user firstly defines specific characteristics of his/her profile (health status, interested activities, etc.) and of involved persons' profile (e.g. mother and young child, father and pregnant daughter, etc.); the hackAIR Problem Description Language (PDL) module will transform the information defined by the user into relevant ontology entities and, together with observation air pollutant values for the location of interest, the ontology-based reasoning module will handle the request as needed. • CQ50. *Which users are considered in one single request?* One direct-user (individual) and none or up to one indirect user (additional person/profiles of interest).

	• CQ51. *Are the aforementioned users contained in one single request?* Yes, they can be considered in one request. This way, the direct user can get recommendation(s) not only for his/her profile but also for other persons of his/her interest; for the latter, relevant hackAIR user profiles should be described through the system. • CQ52. *What is the geographic area associated to the request?* Requests can be made and recommendations will be provided at a city level, or even more local, depending on the spatial resolution of the provided data. • CQ53. *Do air quality observations are given separately or combined?* Air quality data are fused into one single numerical value, for the location of interest. • CQ54. *Which types of data are relevant to a request?* User(s) profile data, user(s) location, activities of interest, air pollutant measurement from fused data for the location of interest. • CQ55. *Do the requests for direct and indirect user are handled as one?* Every request is handled as one, but recommendation(s) are returned separately for every single user profile. • CQ56. *Does the request have any focus on specific environmental type?* Yes, in particulate matter (PM) or aerosol optical depth (AOD) data provided by the Data Fusion Module for the location of interest.
	CQs Group 6: Recommendation data
	• CQ57. *Which data are relevant or can affect the decision inferred by the system?* Age, health status, environmental condition to the requested location, preferred/requested outdoor activities. • CQ58. *What types of recommendations the system will support?* (i) General tips of the day for motivating a change towards more pro-active and environmental friendly behaviour and (ii) personalised recommendations related to user's personal profile for limiting exposure to hazardous AQ condition. • CQ59. *What is the nature of tips of the day?* Alternative ways to trigger behavioural change and to limit potential non-ecological activities will be suggested to the users, through tips of the day. • CQ60. *What is the nature of personalised recommendations?* Suggestions with respect to the user's health status and linked with different types of preferred activities (walking, eating outside, jogging, etc.), will be provided to the users through personalised recommendations. • CQ61. *What is the message associated to the recommendation provided by the system?* "You should reduce prolonged outdoor activity", "You may postpone the picnic for another day", "Ideal air quality for working outdoors", …
7	Pre-Glossary of Terms
	a. Terms from Competency Questions & their Frequency

	user (or users)	31	type	5
	PM	12	hackAIR system	4
	data	10	air quality	4
	request	9	user profile	4
	numerical value	8	indirect user	4
	index	8	categories	5
	age	7	environmental data	3
	environmental	6	recommendation	3
	b. Terms from Competency Questions Answers & their Frequency			
	user (or users)	24	walking	4
	data	13	pregnant	4
	health	10	personalised recommendations	3
	activity (or activities)	9	health status	3
	air	9	air pollutant	3
	profile	8	direct user	3
	location	8	working outdoors	3
	request (or requests)	8	elderly	3
	outdoor	7	measurements	3
	working	6	tips	2
	recommendation	5	system	2
	air quality	4	AOD	2
	sports	4	sensitive health	2
	picnic	4	outdoor activities	2
	outdoors	4	elderly people	2
	c. Objects			
	user, data, request, air quality, air pollutant, health, activity, recommendation, environmental data, measurement, sports, tips, system, …			

3.1.2 Domains of interest

To accurately define the domains of interest that the hackAIR ontology should conceptualise, we took into consideration:

- details specified in the hackAIR use cases [hackAIR D2.2, 2016] - the hackAIR scenarios define two pilot use cases with different personas (end-users that act as the main characters in a scenario) and different characteristics/requirements, the details of which specify efficiently the main entities that should be covered by the ontology schema;
- information derived from the ORSD analysis (intended uses, CQs and their answers) - the ORSD contains all detailed requirements of such representation.

The ontological representation has to be as much comprehensive as possible and to adequately cover the hackAIR use case domains and requirements of interest. From these, the following domains of interest were derived:

- Environmental and AQ monitoring, covering concepts like air pollutant types, observation values, units of measurement, air quality indexes (AQI) and air quality levels, etc.
- User profile information, covering concepts like gender, age, preferences, etc.
- Human health, covering mostly environmental affected and caused diseases, like asthma, cardiovascular diseases, allergic symptoms, etc.
- Human activities, meaning any type of outdoor activities that people perform on a daily basis, like exercising, or even working, travelling, etc.
- Location, considering information about an area of interest, like geographical coordinates, names of cities, countries, etc.
- Recommendation and Decision Support, covering concepts from pure texts to entities for structuring the problem at hand (request for support, recommendation provided, etc.).

3.1.3 Existing ontologies and standards

Ontology reuse is the process of adopting and efficiently integrating available ontological knowledge when developing a new ontology. It is generally considered as a key factor for developing cost-effective, high-quality and interoperable ontologies, since (a) it avoids “re-inventing the wheel”, i.e. rebuilding existing ontologies and resources from scratch, and (b) takes advantage of already formalized ways of representing specific entities in domains of interest. However, ontology engineers have to face the challenges that new ontologies might partially be aligned with the full potential of existing domain-relevant knowledge sources [Bontas et al., 2005].

In the phase of designing and building the hackAIR ontology, requirements are subjected to constant refinements in order to ensure that it adequately covers the knowledge that the ontology is expected to capture. We selected and briefly summarise in the following sub-sections, some of the most motivated existing ontologies and standards that address parts of the hackAIR ontology requirements and of domains of interest. We describe the context of each external module and we refer those classes and properties that could be adopted/specialised/served as building blocks in our ontology, to model information specific to the hackAIR KB and RS. We preferred to adopt and extend parts of existing ontologies and patterns, and not their schema as a whole, since most of the times the external modules diverge from the hackAIR ontology requirements: either only few parts of the external schema are relevant to our context, or some other parts are extensively analysed without adding any value in our ontology semantics but on the contrary affecting significantly the size of the final outcome. Section 3.4.3 presents the exact mapping between existing and implemented concepts in the hackAIR ontology.

PESCaDO

The PESCaDO ontology [Rospocher, 2010; Rospocher, 2014] is a modular domain-ontology exploited for personalized environmental decision support that enables to formally describe: (i) the user decision support request, (ii) the environmental data relevant to process the request, as well as (iii) the decisions and conclusions to be produced. As the ontology conforms to the scope and the basic requirements of the hackAIR ontology, it can be broadly adopted in several concepts of our domains of interest. More specifically, concepts like `pescadoPDL:Activity` [5], `pescadoPDL:User`, `pescadoPDL:Request`, `pescadoDiseases:Disease` [6], `pescadoData:EnvironmentalData` [7], `pescadoData:IndexValue`, etc. align to concepts aimed to be represented in the hackAIR ontology.

FOAF

The FOAF (Friend-Of-A-Friend) ontology [Brickley and Miller, 2014] describes *persons*, *activities* and their *relations* to other people and objects. FOAF integrates three kinds of network: (i) social networks of human collaboration, friendship and association, (ii) representational networks that describe a simplified view of a cartoon universe in factual terms, and (iii) information networks that use Web-based linking to share independently published descriptions of this inter-connected world. Main FOAF terms that can potentially be adopted by the hackAIR ontology are classes `foaf:Person`[8], `foaf:Organization`, and properties `foaf:name`, `foaf:age`.

Dublin Core

The Dublin Core Schema[9] is a small set of vocabulary terms that can be used to describe several kinds of resources. Dublin Core Metadata may be used for multiple purposes, from simple resource description, to combining metadata vocabularies of different metadata standards, and to providing interoperability for metadata vocabularies in the Linked Data cloud and Semantic Web implementations. Metadata entities that can be potentially applicable to the hackAIR concepts are classes `dce:Agent` [10], `dce:Location`, and properties `dce:description`, `dce:date`, `dce:identifier`, etc.

PROV-O

The PROV ontology (PROV-O) [Missier et al., 2013] provides a set of classes, properties, and restrictions that can be used to represent and interchange provenance information generated in different systems and under different contexts[11]. It can be specialized to create new classes and properties to model provenance information for different applications and domains. PROV-O entities that are of special interest within the context of the hackAIR ontology are: `prov:Agent`[12] (and its subclasses `prov:Person` and `prov:Organization`), `prov:Activity`, `prov:Location`, as well as their involved properties.

SWEET

SWEET (Semantic Web for Earth and Environmental Terminology) is an ontology developed by NASA's Jet Propulsion Laboratory, which describes relevant knowledge in Earth Science and Environmental domains [Raskin and Pan, 2003].

---

[5] `pescadoPDL:` is the prefix of the URI <http://www.pescado-project.eu/ontology/pescadoPDL.owl#>
[6] `pescadoDiseases:` is the prefix of the URI <http://www.pescado-project.eu/ontology/pescadoDiseases.owl#>
[7] `pescadoData:` is the prefix of the URI <http://www.pescado-project.eu/ontology/pescadoData.owl#>
[8] `foaf:` is the prefix of the URI <http://xmlns.com/foaf/0.1/>
[9] Available at: http://dublincore.org/documents/dcmi-terms/
[10] `dce:` is the prefix of the URI <http://purl.org/dc/elements/1.1/>
[11] Available at: https://www.w3.org/TR/prov-o/
[12] `prov:` is the prefix of the URI <http://www.w3.org/ns/prov#>

The ontology is highly modular with 6,000 concepts in 200 separate ontologies. It is structured in modules arranged hierarchically, covering from more abstract to more applied concepts; these include a great variety of representations of physical and ecological phenomena, meteorological conditions, processes, activities, etc. Within the hackAIR context, we consider meteorological relevant concepts as most relevant to the domains of interest. Some identical concepts (grouped into modules) of the latest version (v2.3) of SWEET[13] are given below:

- The *Atmosphere* module[14] contains different atmospheric and meteorological phenomena, such as *sky* condition (`ClearSky`, `CloudCover`, `PartlySunny`, `PartlyCloudy`, etc.), *weather* condition, *precipitation* (`Drizzle`, `Rain`, `Snowfall`, `Storm`, etc.), and many more.
- The *PhysicalProperty* module and its sub-classes describe, among others, concepts like *pressure*, *temperature*, *speed* as well as *height*, *population* and *size*.
- The *Units* module[15] contains concepts for representing units of measurement, covering: simple units (*kilo, micro, milli, nano, meter, second,* etc.), units defined by product (*kilogramPerMeterSquared*, *meterPerKelvin, etc.*), units defined by scaling (*millimeter*, *nanometer*, *degrees*, *percent*, *hour*, *minute, day*, etc.), units defined by raising to power (*perMeter*, *perMeterSquared*, *perKilogram*, *perSecond*, etc.), and many more. Each definition contains the symbol description (property `hasSymbol`), the numerical value of the scale (property `hasNumericalValue`), the operands of their calculation function (property `hasOperant`) and other descriptive details.
- The *EnvironmentalImpact* module provides descriptions for concepts related to *air quality*[16] (`AirQuality` – subclass of `AirPollution` – subclass of `Pollution`, `Smog`, `BrownCloud`, etc.), *air pollutants*[17] (among them, both `PM10` and $PM_{2.5}$, ozone – O_3, nitrogen oxides – `NO` and NO_2, sulphur dioxide – SO_2), *environmental indicators*[18] (`AirQualityIndex` – subclass of `EnvironmentalIndex`).
- The *EnvironmentalStandards* module[19] provides definitions of different environmental related standards: e.g. `AcceptableLevel`, `AirQualityStandards`, `AmbientAirQualityStandards`, `NationalAmbientAirQualityStandards`, etc.

GeoNames Ontology

The GeoNames ontology[20] provides elements of description for geographical features defined in the GeoNames database, which contains over 10,000,000 geographical names corresponding to over 7,500,000 unique features. The GeoNames toponyms have a unique URL with a corresponding RDF web service that allows accessing the geospatial semantic information attached to them. Concepts like *country*, *capital*, *population*, *languages*, *city*, etc. are covered with both structural (schema) and actual (instances) details, which could be of special interest for current and future functionalities of the hackAIR ontology. Details for every interested city/country could be potentially derived through the use of the GeoNames services.

LinkedGeoData

[13] Available at: https://sweet.jpl.nasa.gov/sweet2.3
[14] Available at: https://sweet.jpl.nasa.gov/filebrowser/download/18689
[15] Available at: https://sweet.jpl.nasa.gov/filebrowser/download/18835
[16] Available at: https://sweet.jpl.nasa.gov/filebrowser/download/18707
[17] Available at: https://sweet.jpl.nasa.gov/filebrowser/download/18665, https://sweet.jpl.nasa.gov/filebrowser/download/18670
[18] Available at: https://sweet.jpl.nasa.gov/filebrowser/download/18751
[19] Available at: https://sweet.jpl.nasa.gov/filebrowser/download/18658
[20] http://www.geonames.org/ontology/documentation.html

The LinkedGeoData knowledge base[21] [Stadler et al., 2012] provides a rich integrated and interlinked geographic dataset for the Semantic Web. The majority of their data is obtained by utilising freely available spatial data from OpenStreetMap[22], which is a collaborative project to create a free editable map of the whole world. Among others, concepts of the LinkedGeoData ontology enable the explicit discretization of common concepts `City`/`Country`, by providing, for example, notions like `Village`, `NationalPark`, `Shop`, `Cafe`, `ResidentialHighway`, etc. It could be useful to adopt such notions in the hackAIR ontology, in case that we want to handle location-observation information in more detail – not only at a city/country level.

DOLCE

Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) [Gangemi et al., 2002] aims at capturing the ontological categories underlying natural language and human common sense. DOLCE does not commit to a strictly referentialist metaphysics related to the intrinsic nature of the world. Rather, the introduced categories are thought of as cognitive artifacts, which are ultimately depending on human perception, cultural imprints and social conventions. DOLCE definitions represent abstract concepts that could be utilised in any domain-specific ontology. Entities like *PhysicalRegion* (i.e. physical space, area, etc.), *AgentivePhysicalObject* (i.e. a human person), *State* (i.e. being sitting, being happy, doing something, etc.), *AmountOfMatter* (i.e. quantity of something), etc. could be of interest to the hackAIR domains context as well.

Ontology Design Patterns

Ontology Design Patterns (ODPs) [Gangemi and Presuttin, 2009] are components, methods and ontology snippets that support pattern-based ontology design. Several ODPs have been examined while designing the hackAIR ontological framework. Indicatively:

- Place[23] – this pattern is aimed at representing the location of an entity, which might be physical, partial, metaphorical, social, etc.
- Action[24] – this pattern is to model actions (i.e. activities) that are proposed, planned and performed or abandoned, together with their status and duration in time.
- Activity reasoning[25] – it incorporates two perspectives of activities: a workflow perspective, which is often observed in planning-related applications, and a spatiotemporal perspective, which is often found in geographic activity analysis.
- PartOf[26] – this pattern addresses the mereology of entities; it is used to represent entities and their parts, i.e. part-whole relations, with transitivity.
- Class Union[27] – this pattern represents cases where a class denoted in an ontology is the union of two classes in another ontology. This pattern is agnostic as to whether the correspondence is unidirectional or bidirectional.
- EventProcessing[28] – this pattern intends to model event objects, their attributes, and their relations actual events, and sensor readings producing the events. Different types of event objects, such as complex,

[21] http://linkedgeodata.org/About
[22] http://www.openstreetmap.org/
[23] http://ontologydesignpatterns.org/wiki/Submissions:Place
[24] http://ontologydesignpatterns.org/wiki/Submissions:Action
[25] http://ontologydesignpatterns.org/wiki/Submissions:An_Ontology_Design_Pattern_for_Activity_Reasoning
[26] http://ontologydesignpatterns.org/wiki/Submissions:PartOf
[27] http://ontologydesignpatterns.org/wiki/Submissions:Class_Union
[28] http://ontologydesignpatterns.org/wiki/Submissions:EventProcessing

composite, and simple events are modelled, properties for expressing relations between event objects, such as constituency and componence are expressed, and attributes of event objects, such as timestamps and other data values.

- Policy[29] – the pattern intends to model policies, their characteristics and their associated entities, such as processes and agents.
- Observation[30] – the intent of this pattern is to represent observations of things, under a set of parameters. Common parameters may be the time and place of the observation, but may also be any feature that is observed concerning the specific observed entity.

3.2 Technologies and Tools

3.2.1 Ontology language

The ontology language deployed for developing the hackAIR ontologies is **OWL 2** (Web Ontology Language), an ontology language for the Semantic Web with formally defined meaning. It was published by *the W3C OWL Working Group* in 2012 [W3C, 2012a] and is now a W3C recommendation. OWL 2 is based on a strong mathematical background (i.e. Description Logics[31]).

OWL 2 ontologies can be used along with information written in RDF, and themselves are primarily exchanged as RDF documents. Data is structured in **RDF triples**, which, in other words, are statements in the form ***<subject predicate object>***. Each entity within a triple is associated with a uniform resource identifier (URI) usually in the form of http:// address, which is a unique identification that serves the principles of the Semantic Web. URIs involve two parts:

- **the base URI** (the leftmost) part of the URI which is common across multiple entities in a specific ontology. Since URIs can be long, a short form (**prefix**) can be specified to represent the commonly used part of the URI.
- **the URI fragment** part which is the part of the URI after a delimiter (usually #). This part denotes a recognizable name for the described entity which should follow the basic rules and guidelines for naming and labelling ontologies [Noy and McGuinness, 2001].

OWL 2 offers the following essential modelling building blocks [W3C, 2012b]:

- **Classes** - an abstraction mechanism for grouping objects with similar characteristics and denote the set of objects comprised by a concept. There may be diverse criteria for grouping objects/individuals and also one individual may simultaneously belong to several classes. Classes can also form a hierarchy of more generic (super-classes) and more specific (sub-classes) notions.
- **Individuals** - the objects belonging to one or more classes and are also referred to as the classes' instances. Individuals are formed by taking into account things that are abstract (classes) and establishing values so that they become real assertions.
- **Properties** - indicators of existing relationships between entities. *Domain* and *Range* are two key notions that restrict the type of entity to be used on each side of a relationship. Properties are further classified as:
  - **Object properties**, which describe how classes and their individuals can be related to each other;
  - **Data properties**, which attribute data values to individuals, either using default data types (e.g. string, integer, boolean, etc.) or within pre-defined data range expressions;

[29] http://ontologydesignpatterns.org/wiki/Submissions:Policy
[30] http://ontologydesignpatterns.org/wiki/Submissions:Observation
[31] https://en.wikipedia.org/wiki/Description_logic

- **Annotation properties**, which give additional description to the domain being modelled, without having any effect on the logical aspects of the ontology.

3.2.2 Third-party frameworks

The following third-party software tools and frameworks were used during the development and utilisation of the hackAIR ontologies:

- **TopBraid Composer**[32] **–** it is a visual modelling environment for creating and managing domain models. Its graphical user interface (GUI) enables the fast design and development of ontologies; it offers a convenient drag-and-drop, form-based user interface with the ability to view and edit ontologies in a variety of serialization formats. Moreover, TopBraid Composer seamlessly integrates logical and rule-based reasoning engines, which is of most importance for the development, consistency checking and proper functioning of the hackAIR RS.
- **SPIN – SPARQL Inferencing Notation**[33] – it is a state-of-the-art framework and W3C recommendation for rule-based reasoning in ontologies. The SPIN standard is used to represent SPARQL rules and constraints on ontology-based models. It allows also defining new SPARQL functions and querying templates for reusability and extensibility issues. Reasoners integrated in TopBraid Composer support SPIN inference.
- **Apache Jena**[34] – it is an open source Java framework for developing web-applications and services. It includes: (a) the Ontology API for manipulating ontology models, and (b) the Inference API for reasoning over populated/modelled data stored in the hackAIR ontology-based KB.
- **yEd Graph Editor**[35] **and Graffoo**[36] – yEd is a free general-purpose diagramming program with a multi-document interface. It can be used to draw many different types of diagrams with ease, via an intuitive user interface. Graffoo [Falco et al., 2014] is a graphical framework for ontologies that can be loaded as a separate section in the yEd palette. We use both technologies to visualise information modelled in the hackAIR ontologies with a well-established, recognisable and easily interpretable way.

3.3 Architecture of the hackAIR ontology

The current version of the hackAIR ontology (version 1.0 – June 2017) consists of three interconnected sub-ontologies (see Figure 2):

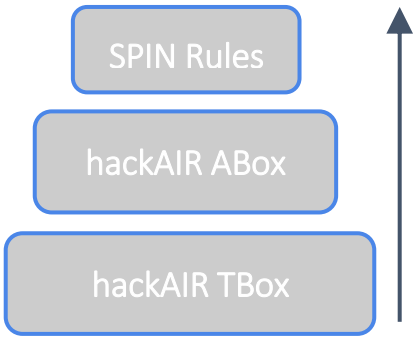


Figure 2 – Architecture of the hackAIR ontology

[32] http://www.topquadrant.com/tools/IDE-topbraid-composer-maestro-edition/
[33] http://spinrdf.org/
[34] https://jena.apache.org/
[35] https://www.yworks.com/products/yed
[36] http://www.essepuntato.it/graffoo

- the **hackAIR TBox**[37] formalises information relevant to the hackAIR concepts (user profile, AQ measurements, recommendations, etc.) in a detailed schema with classes, properties and relations;
- the **hackAIR ABox**[38] formalises information relevant to membership/attribute assertions, i.e. actual users, observations, etc.;
- the **hackAIR SPIN Rules** formalise sets of rules for reasoning (inferencing) low-level derivations (age groups, user groups, AQ levels, etc.) as well as high-level interpretations (user-profile driven recommendations).

The aforementioned sub-ontologies are interconnected, in the sense that the lower layers are imported/adopted by the higher layers; for example the *hackAIR ABox* imports the *hackAIR TBox*, and the *hackAIR SPIN Rules* imports both *hackAIR ABox* and *TBox*.

According to ontology best practices, it is preferred to maintain a relative split between instances (ABox) and the conceptual entities (TBox) which describe the domain of interest [AI^3, 2009]. Within hackAIR, the key reason to follow such a multi-layered approach is to keep concepts and rules in distinct schemas. We selected to distinguish the basic structure (concepts, hierarchy) from instances (individuals) and from dynamic rule-based operations (reasoning and decision support), in order for the overall hackAIR ontology to be more modular, adaptable and easily extensible in any of the three layers, i.e. three different ontology modules.

3.4 Main ontology concepts

In the current section, we thoroughly present the first stable version of the hackAIR ontological framework. We describe in detail the entities that the ontology includes, thus showing how information related to user profiles, AQ observations, requests, recommendations, etc. can be populated in the ontology and how instances are related to each other. The base URI as well as the defined prefix for each of the three sub-ontologies defined in the hackAIR ontological framework is listed in Table 3. Ontologies are not publicly available yet but they will be released by M20, together with the implementation of the 1st hackAIR pilot framework.

Table 3 – hackAIR URIs and prefixes of the ontological framework

prefix	Base URI
`hackairTBox:`	http://mklab.iti.gr/hackair/hackairTBox#
`hackairABox:`	http://mklab.iti.gr/hackair/hackairABox#
`hackairSPIN:`	http://mklab.iti.gr/hackair/hackairSPIN#

It should be noted that for the sake of brevity, whenever a class or property or individual is referred in the text or in graphs with only a single name, within the following sections of the document, then the prefix or the complete URI of the entity is not mentioned; on the contrary, prefix named `hackairTBox:` is implied for the class/property references, prefix named `hackairABox:` is implied for the populated individuals (instances) and prefix named `hackairSPIN:` is implied for the SPIN rules implemented in the ontology.

3.4.1 The hackAIR TBox and ABox

In the current section, we present the main concepts implemented in the hackAIR TBox ontology and we describe the modelling schema (hierarchy, relations) that the ontology supports. We additionally present indicative instantiations

[37] The letter "T" in TBox stands for the terminological knowledge (i.e. terms/schema).
[38] The letter "A" in ABox stands for the assertions (facts).

of the ontology classes, in the form of graphs, in order to describe how main concepts are realised as actual instances in the hackAIR ABox ontology. An excerpt of the taxonomy behind the hackAIR TBox ontology is presented in Figure 3.

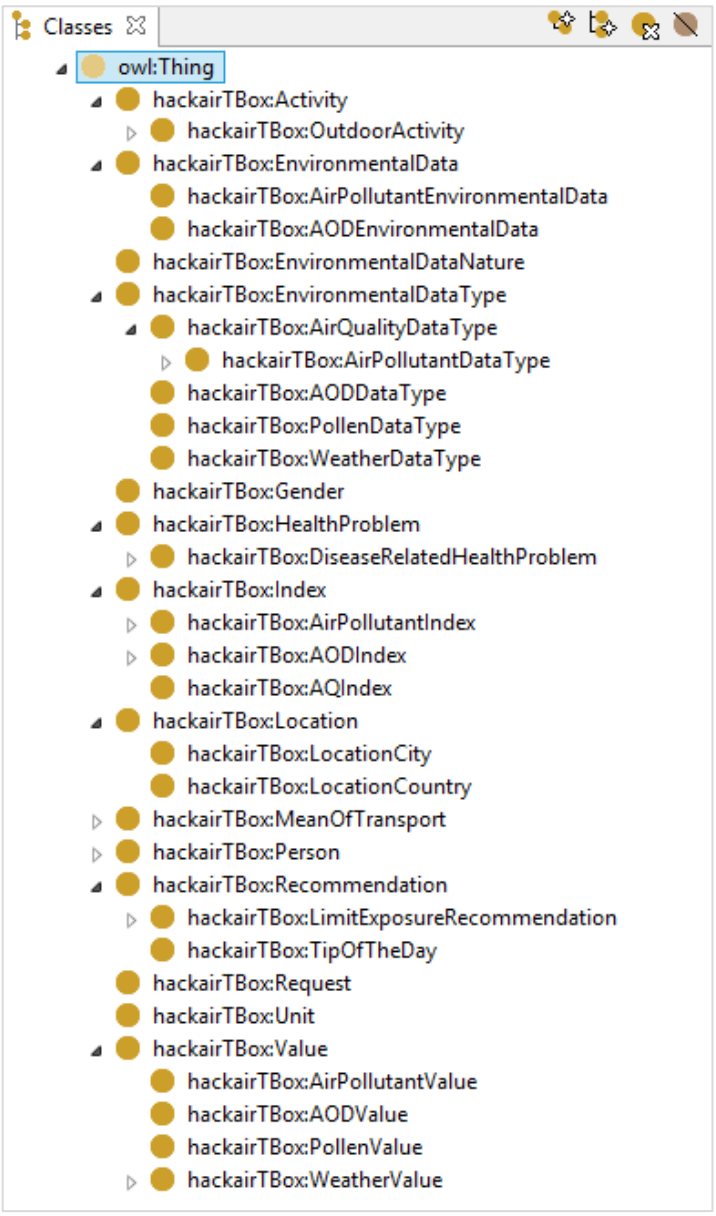


Figure 3 – Excerpt of the hackAIR TBox ontology (screenshot from TopBraid Composer)

Class `Person`

This class can be used to identify details of personal profiles of people involved in the hackAIR system. Assertions on individuals of `Person` type allow representing information about the hackAIR user-profile. Distinct sub-classes of the class `Person` cover different hackAIR user profiles with respect to specific parameters:

- **the direct access/use of the system**: a person can be either a `DirectUser` or an `IndirectUser`;
- **the individual's age**: users can be classified into one of the relevant `Person` sub-classes `InfantPerson`, `ToddlerPerson`, `ChildPerson`, `YoungPerson`, `AdultPerson`, `MiddleAgePerson`, `ElderlyPerson`;
- **the individual's health condition**: users may be classified into classes `PregnantFemalePerson`, `SensitiveHealthPerson`;

- **the defined preferred activities**: individuals of **`Person`** type can be further classified as `OutdoorJobPerson`, `SportsGeneralPerson`, `SportsWalkingPerson` and `SportsPicnicPerson`.

Table 4 – Sub-classes of Person in hackairTBox ontology

hackairTBox class name	Description
`DirectUser`	corresponds to the user of the system
`IndirectUser`	his/her user profile is related to an individual of `DirectUser` type
`InfantPerson`	age less or equal to 1 year old
`ToddlerPerson`	age between (1,3] years old
`ChildPerson`	age between (3,10] years old
`YoungPerson`	age between (10,17] years old
`AdultPerson`	age between (17,40] years old
`MiddleAgePerson`	age between (40,60] years old
`ElderlyPerson`	age more than 60 years old
`PregnantFemalePerson`	female person who is pregnant
`SensitiveHealthPerson`	person with health sensitivities
`OutdoorJobPerson`	person who works outdoors
`SportsGeneralPerson`	person who prefers doing sports (in general)
`SportsWalkingPerson`	person who prefers walking outdoors
`SportsPicnicPerson`	person who prefers eating outside (picnic)
`CombinedCategoriesPerson`[39]	person belonging into combination(s) of Person sub-classes. Relevant sub-classes of this class exist, i.e. `Elderly_OutdoorJob_Person` (person more than 60 years old that declares he/she has a daily outdoor job), `Pregnant_SportsPicnic_Person` (woman that she is pregnant and she prefers to go for picnic outside), etc.

The aforementioned classes of `Person` type implicitly define the types of users that are considered by the hackAIR system as most vulnerable to hazardous AQ conditions, and for which the RS will support the inference of relevant recommendations.

[39] This class is useful for the reasoning task and is described in detail in Section 6

Apparently, an individual of class `Person` may belong to more than one of `Person` sub-classes; for that reason a sub-class named `CombinedCategoriesPerson` is created in the ontology to gather all types of individuals belonging into any of the possible combinations of main sub-categories (see example description in last row of Table 4).

The instantiation of an individual of `Person` type, and thus of the relevant user profile, is feasible with the declaration of the associated object/data properties, as presented in both Table 5 and Figure 4.

Table 5 – Object and data properties that are connected to the class Person

Object property	Domain	Range	Description
`hasGender`	`Person`	`Gender`	gender of the user
`hasLocation`	`Person`	`Location`	default location of the user; considered for potential recommendations
`isSensitiveTo`	`Person`	`HealthProblem`	indicates that the user suffers from a specific health problem; either general or a `DiseaseRelatedHealthProblem`
`hasPreferred Activity`	`Person`	`Activity`	defines one or more activities that the user may defined as preferred ones and for which he/she may receive relevant recommendations
`usesAvailableMOT`	`Person`	`MeanOfTransport`	correlate possible means of transport which individuals of class `Person` use throughout their daily routine[40]
`hasRelatedPerson`	`Person`	`Person`	an individual of type `Person` is linked to one or more individuals of the same type, who are indirect users of the system
`isProvidedWith Recommendation`	`Person`	`Recommendation`	association between a user and a derived recommendation from the hackAIR RS
Data property	**Domain**	**Range**	**Description**
`hasName`	`Person` ∪ `Location`[41]	`xsd:string`	name of the user

[40] This information is not directly asked to the user; on the contrary it is implied, if user defines an activity that involves a mean of transport (e.g. biking activity implies the use of bike)

[41] the symbol ∪ is used to denote the union of two classes, i.e. the `hasName` property can have as domain either the `Person` or the `Location` class.

hasAge	Person	xsd:integer	the age of the user
isPregnant	Person	xsd:boolean	true/false if a female user declares that she is pregnant
worksOutdoors	Person	xsd:boolean	true/false if the user declares that he/she has an outdoor job. The true value can be assigned automatically if the user defines WorkingActivity as a preferred activity in his/her profile.
isDoingSports	Person	xsd:boolean	true/false if the user states that he/she is interested in doing sports (in general). The true value can be assigned automatically if the user defines SportsGeneralActivity as a preferred activity in his/her profile.
belongsTo SensitiveGroup	Person	xsd:boolean	true/false if the user defines that he/she has a general health sensitivity

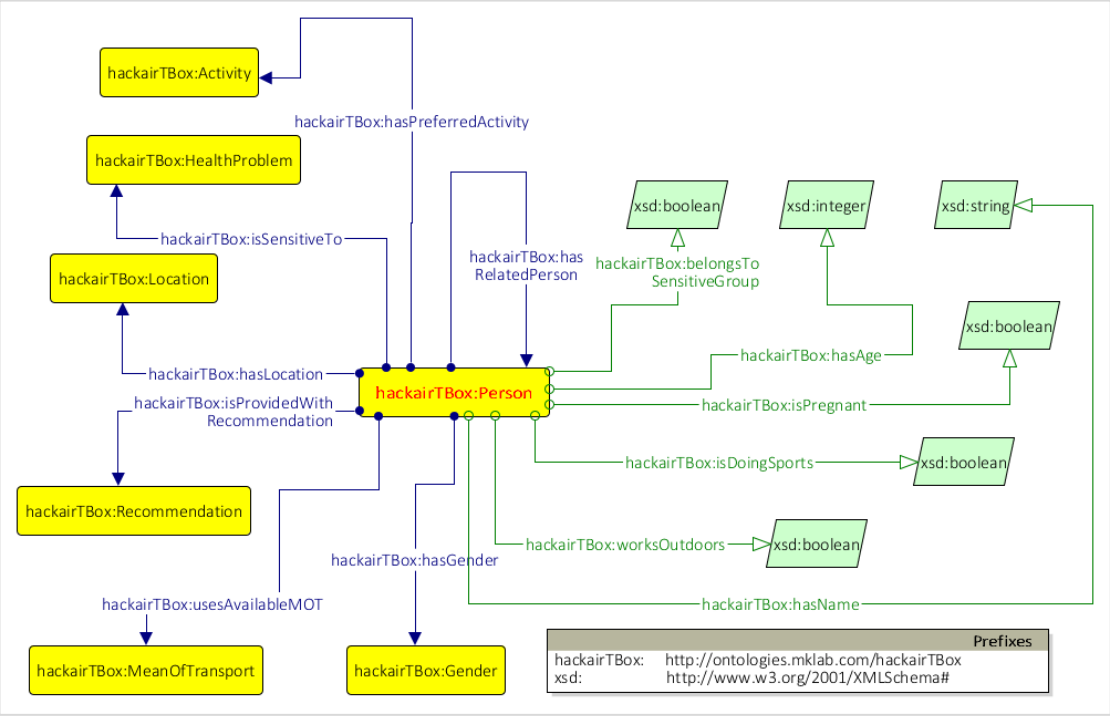


Figure 4 – Available assertions on individuals of the class Person

Class Activity

This class can be used to represent the possible activities users can specify as preferred activities in their hackAIR profile. The class `Activity` is further divided into:

- `OutdoorActivity`, which includes all *hackAIR system* -specified activities that are performed **outdoors**, as given in its following sub-classes:
  - `JoggingActivity`
  - `PlayingInParkActivity`
  - `TennisActivity`
  - `SwimmingOutdoorsActivity`
  - `MovingByOpenMOTActivity`, meaning the activity of mobility by using a specific MOT while being exposed in open air (e.g. by bike, motorcycle, etc.)
  - `SportsGeneralActivity`
  - `WorkingActivity`
  - `WalkingActivity`
  - `PicnicActivity`

One important aspect is that a person can define one or more preferred activities in his/her hackAIR profile; the RS will then provide one relevant recommendation for every specified activity in user's profile. It should be additionally noted that in the current version of the hackAIR ontology and of the DSS capabilities, all activities are considered as of `SportsGeneralActivity` type, except from activities related to *working*, *walking* or eating outdoors (*picnic*), for which *specialised activity-related recommendations* have been specified (for more details, see Section 6.1.2).

Class `HealthProblem`

The individuals of this class represent possible health problems the users may specify into their hackAIR profile. The `HealthProblem` class is divided into:

- `DiseaseRelatedHealthProblem`, which includes individuals related to cardiovascular, circulatory or respiratory diseases.

Though shallow, the discretisation of the aforementioned class is due to the fact that we should avoid recording any sensitive information from the user, such as the health status. Hence, in the ontology population process we either create a link between an individual of `Person` type with an individual of `HealthProblem` type, in general, or a boolean value of the `belongsToSensitiveGroup` object property will be sufficient. In both cases the RS will acknowledge the semantics behind these relations and provide the user with a recommendation relevant to his/her sensitivity under poor air quality conditions.

Class `Location`

Concerning the location of the user, the hackAIR ontology and thus the RS will track/utilise only information about the city and country of the area of interest. The `Location` class describes the default location of the user, with additional use of its sub-classes named `LocationCity` and `LocationCountry`. An individual of type `Location` (and thus of its sub-classes) may be linked to other individuals/values with the use of specific object/data properties (see details in Table 6).

Table 6 – Object and data properties that are connected to the class Location

Object property	Domain	Range	Description
hasEnvironmentalData	Location	Environmental Data	an individual of EnvironmentalData type which corresponds to the actual/aggregated measurement for specific AQ parameter, can be assigned to an individual of Location type
hasRelatedIndex	Location	Index	an individual of Location type is linked to an individual of Index type, according to the numerical value of the EnvironmentalData individual
Data property	**Domain**	**Range**	**Description**
hasName	Location ∪ Person	xsd:string	name of the location, either of the city or of the country

Individuals of class LocationCity and LocationCountry are linked between them via the properties hasCountry (domain: LocationCity, range: LocationCountry, inverseProperty: isCountryOf) and hasCity (domain: LocationCountry, range: LocationCity, inverseProperty: isCityOf).

Class MeanOfTransport

An individual of MeanOfTransport class could be any public or private mean of travel modality, i.e. bike, car, train, tram, metro, bus, etc. Such individuals can be attached to:

- instances of type Person via the property usesAvailableMOT (sub-property of availableMOT), so as to define one or more preferred means of transport that the user uses throughout his/her daily routine;
- instances of type Activity (outdoor activity) via the property modalityMean, so as to define the mean by which the mobility activity (MovingByOpenMOTActivity) is performed.

A detailed depiction of such concepts and relations is given in Figure 5.

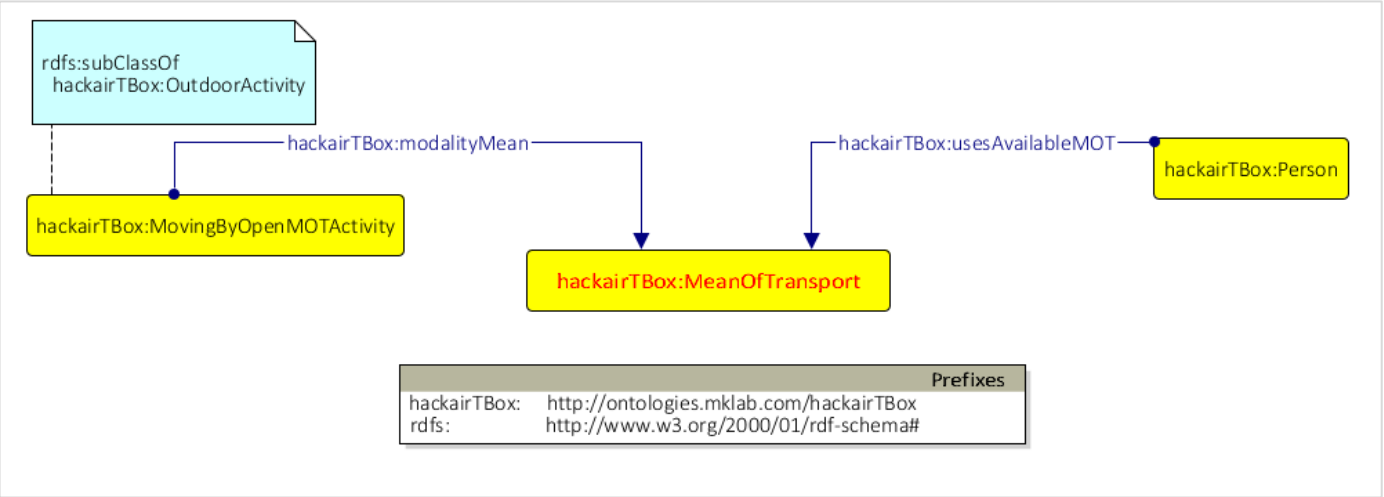


Figure 5 – Available assertions on an individual of MeanOfTransport type

In the current version of the ontology, we focus only on those means of transport that are related to activities, e.g. bike and biking activity, motor and motorcycling activity, etc. We present the relation between `Person` and `MeanOfTransport` only for semantic purposes; such a relation (if exists) is implying from the predefined activities, e.g. if someone selects `biking_activity` as preferred one, then a relation like <`person_x usesAvailableMOT bike`> is defined in the ontology. Potentially, information related to public/private MOT could be used for providing alternative recommendations to the users regarding their mean of mobility (e.g. prefer moving by bus instead of bike when AQ is hazardous) in future versions of the RS. Such a utility requires the request of user for which type of MOT he/she owns/uses, or even more, the knowledge and recording of all available public MOT in the ontology, for any location (city) of interest, in order to suggest alternative MOT that eliminates the outdoor exposure or the production of pollution.

Class `EnvironmentalData`

This class represents all environmental related measurements that are considered in the hackAIR system, meaning observations from sensors, monitoring stations, AQ related values from fused or forecasted data or from sky-depicted images, etc. The main definitions of the linked properties and classes were adopted from the PESCaDO-Environmental Data Ontology [Rospocher, 2010; Rospocher, 2014]. Assertions that can be made on individuals of the described class are mentioned in Table 7.

Table 7 – Object properties that are connected to the class EnvironmentalData

Object property	Domain	Range	Description
`hasEnvironmental DataType`	`Environmental Data`	`Environmental DataType`	connects the individual of class `EnvironmentalData` with the type of data that it represents
`hasEnvironmental DataNature`	`Environmental Data`	`Environmental DataNature`	creates a reference to the nature of the environmental data (*observed*, *forecasted*, *fused*, etc.)

hasNumericalValue	EnvironmentalData	Value	the actual measurement (numerical value) of the environmental data
hasRelatedIndex	EnvironmentalData ∪ Location[42]	Index	links an individual of EnvironmentalData to a relevant instance of AQ related Index

Class EnvironmentalDataType

The individuals of this class represent all possible types of environmental data that are considered in the hackAIR system. The individuals are further categorised in sub-classes:

- AirQualityDataType, which includes as sub-class the AirPollutantDataType type;
- AODDataType, the individuals of which will be relevant to the AOD measurements provided by the hackAIR system;
- PollenDataType, for pollen related measurements (designed not to be taken into account in the current version of the hackAIR system);
- WeatherDataType, for weather related measurements, i.e. wind, precipitation, temperature, etc. (designed not to be taken into account in the current version of the hackAIR system).

Class AirPollutantDataType

This class, as being a sub-class of AirQualityDataType and of EnvironmentalDataType, it represents individuals of specialised AQ related type. Individuals of class AirPollutant could be any relevant air pollutant that is monitored in the hackAIR system. Thus, individuals of AirPollutant will represent specific types of particulate matter (i.e. PM_{10} and $PM_{2.5}$).

Class EnvironmentalDataNature

This class is used to represent the nature (i.e. observed, forecasted, fused, historical, etc.) of EnvironmentalData. Every individual of EnvironmentalData should have exactly one EnvironmentalDataNature assertion.

Class Index

This class can be used to indicate a qualitative value of a quantifiable entity, like e.g. index levels for air pollutant measurements. Such qualitative values are represented as individuals of class Index and they correspond to specific ranges of their relevant quantitative measurements; for example, a value of PM_{10} between [0-20] $\mu g/m^3$ corresponds to and index of *very good* AQ condition.

The Index class is further divided into sub-classes: AirPollutantIndex, AODIndex and AirQualityIndex, for all of which relevant individuals are initialised in hackAIR ontology that represent the *very good*, *good*, *medium* and *bad* AQ condition. The property hasIndexValue targets to the relevant entry of xsd:string type that defines the aforementioned index values. More details on AQ indexes and their corresponding scales are given in Section 6.3.1.2, while an empirical matching between AQ indices and qualitative levels of air pollutants is given in Table 8.

[42] An individual of Location type could also be linked to a relevant Index via the property hasRelatedIndex, to represent the same relation, i.e. that a specific location has an AQ level of type Y, where Y is the specific AQ index (very good, medium, good, bad)

Table 8 – Empirical matching between air pollutant indices and air pollutant levels

Index	Pollutants' level
`very good`	very low
`good`	low
`medium`	moderate
`bad`	high / very high

Class `Value`

This class represents quantitative values of hackAIR entities that are measurable. Any type of `EnvironmentalData` (defined by `EnvironmentalDataType`) that has specific numerical measurement can be instantiated via the class `Value`; thus, relevant sub-classes exist in the ontology, like for example: `AirPollutantValue`, `PollenValue`, `AODValue`, `WeatherValue` (`RainValue`, `HumidityValue`, `WindSpeedValue`), etc.

Individuals of class `Value` may have the following assertions:

- `hasUnit`, which connects a specific value with its corresponding unit of measurement;
- `hasValueValue`, which defines the actual numerical (`xsd:double`) value associated to the individual.

Class `Unit`

This class encapsulates information about the units of measurement of hackAIR related entities. Every numerical value of type `Value` should be asserted with an individual of type `Unit`. The data property that describes the unit symbol of the individual of `Unit` class is named `hasUnitSymbol` and targets to a value of `xsd:string` type. Example target values instantiated in hackAIR ABox, are given in Table 9:

Table 9 – Excerpt of official units of measurement[43] that are represented in hackAIR ontology

Unit individual	Unit symbol (`hasUnitSymbol`)	Target measurement
microGramsPerCubicMeter	"µg/m3"^^xsd:string	PM concentration
grainsPerCubicMeter	"grains/m3"^^xsd:string	Pollen count
degreesC	"C"^^xsd:string	Temperature in degrees Celsius
second	"s"^^xsd:string	Time is seconds
kilometresPerHour	"km/h"^^xsd:string	Wind speed

[43] Defined by the SI – International System of Units, available at: http://www.bipm.org/en/measurement-units/

percent	"% "^^xsd:string	Percentage (number or ratio expressed as a fraction of 100); useful for declaring Humidity or Precipitation values

Class `Request`

This class represents the actual user requests expressed according to the hackAIR problem description entities (more details in Section 4.2); a request could be submitted by the user, through the use of the hackAIR platform and its represented information will be of significance in order to provide representative/personalised recommendations. The aforementioned class is further divided into sub-classes that correspond to the nature of the request, i.e. querying for the tip of the day, requesting for advice that takes into account a health issue, or preferred activity (-ies), or querying for decision support of the primary (direct) user or of the involved (indirect) users.

An individual of `Request` type can be instantiated for:

- *one* **specific user profile** with its additional linked profile (if any),
- *one* **defined location,**
- *one* **specific (current) AQ observation**, and
- *none/one/more than one* **preferred user activities.**

The individual of the aforementioned type can be linked to other individuals of the hackAIR ABox via the following object properties:

- `involvesPerson`: targets to the individual of class `Person` that specifies the user and his/her profile;
- `involvesLocation`: targets to the individual of class `Location`, which specifies the current location of the user;
- `involvesEnvironmentalData`: targets to the individual of class `EnvironmentalData` that specifies the actual numerical observation of air pollution for the location of interest;
- `involvesActivity`: targets to one or more individuals of class `Activity` that specifies the type of activities for which the user desires to get a relevant recommendations.

The individual of type `Request` can have the following datatype property assertions:

- `hasSubmissionDateTime`: every request can have exactly one of this declarations, which indicates the date/time (`xsd:dateTime`) when the request is submitted by the user.

Class `Recommendation`

This class formalises information about the notion of recommendations that are generated by the DSS according to user's profile and his/her request. The `Recommendation` class is divided into two sub-classes, which are in line with the supported types of recommendation by the hackAIR DSS. These are:

- `TipOfTheDay`: individuals of this class represent specific messages in the form of small tips that are potentially helpful knowledge for reducing the production of pollution or improving he ambient air quality, through behavioural change within everyday life.
- `LimitExposureRecommendation`: individuals of this class contain specific messages that will be provided to the user in the form of short advice with a more personalised expression. Texts contained on individuals of such type reflect to the characteristics of the user profile (age, health sensitivity, preferred activities) and of the existing AQ conditions, and usually are suggestions for limiting the exposure to air pollution.

An individual of type `Recommendation` can have the following property assertions (Table 10):

Table 10 – Object and data properties that are connected to the class Recommendation

Object property	Domain	Range	Description
`isAssignedTo PersonCategory`	`Recommendation`	`Person` ∪ `Combined CategoriesPerson`	each recommendation is assigned into one or more individuals of target (Range) class
`isAssignedTo LevelOfAQ`	`LimitExposure Recommendation`[44]	`Index`	each recommendation of `LimitExposure Recommendation` type is assigned to specific individuals AQ index type
Data property	**Domain**	**Range**	**Description**
`hasDescription`	`Recommendation`	`xsd:string`	contains the actual text of the `Recommendation` that is provided to the user as advice

The aforementioned representation plays a significant role in the DSS; rules (see Section 6.3) implemented on top of the ontology infer automatically the appropriate individual of `Recommendation` type to the user(s) that is/are involved in a `Request`, with respect to:

- the classes/sub-classes of `Person` type where the user belongs to, according to his/her profile specifications;
- the value of individuals of `Index` type where the individual of `EnvironmentalData` type is categorized, according to the recorded/observed numerical value of the latter.

In the current version of the hackAIR ontology a sufficient number of recommendations has been specified as individuals of both sub-classes of `Recommendation` type. The list of all defined tips of the day in the hackAIR ontology is given in Appendix 8.1, while the corresponding list of all defined recommendations per different AQ condition and different user profile is given in Appendix 8.2. An indicative example of recommendation of `LimitExposureRecommendation` type is given in Figure 6.

[44] Individuals of `TipOfTheDay` type do not declare the `isAssignedToAQIndex` property, since tips of the day are provided to any direct user (upon request) regardless of the existing AQ condition.

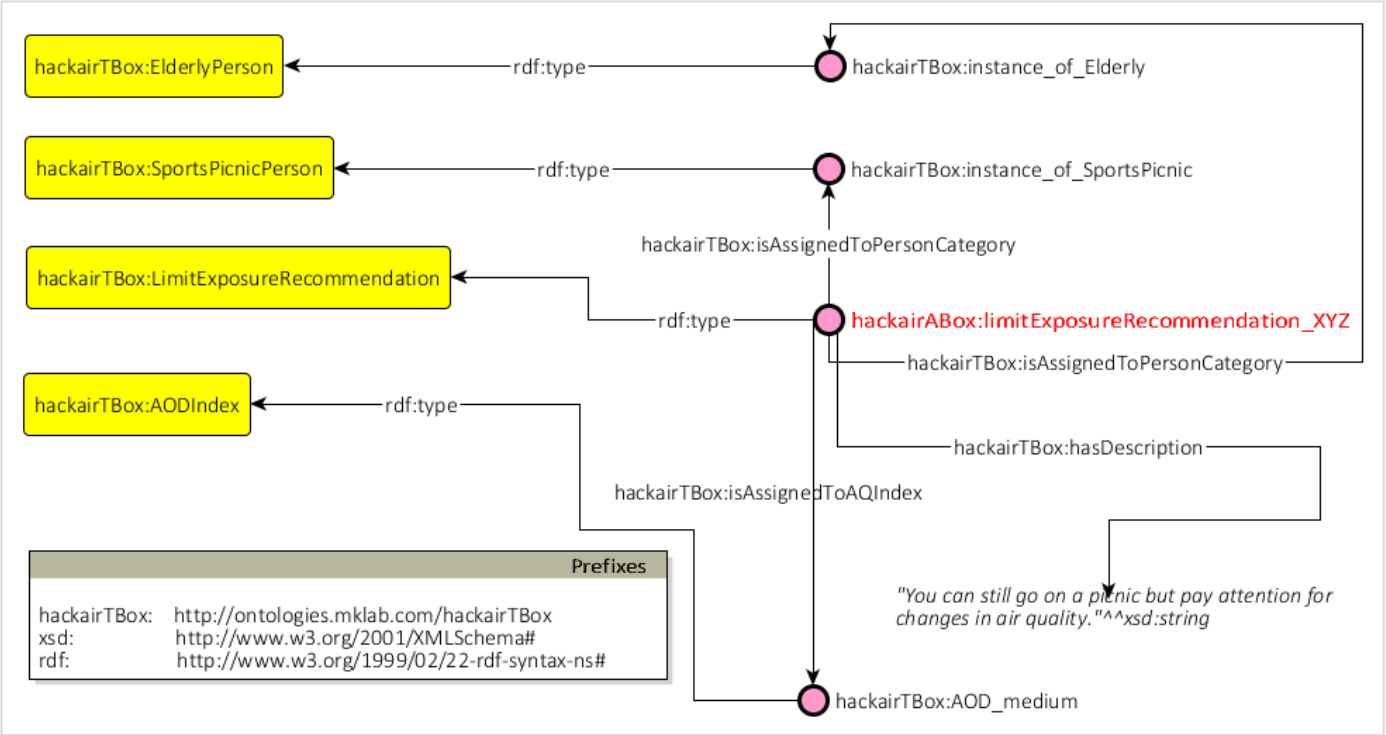


Figure 6 – Instantiation of the class Recommendation, covering a specific case scenario

This visualisation represents the linked classes and relations of a specific hackAIR recommendation that is intended for: an elderly person (age above 60 years old) who declares to the system that he/she is interested to go for a picnic. According to the hackAIR ontology, this person will be an individual of both `hackairTBox:ElderlyPerson` and `hackairTBox:SportsPicnic Person` type. If the user performs a request (`hackairTBox:Request`) to the hackAIR system and, at the same time, the air quality in his/her area is in moderate condition (<`hackairTBox:AOD_medium rdf:type hackairTBox:AODIndex`>) then a specific individual of `hackairTBox:LimitExposureRecommendation` type will be asserted to the user's request for recommendation (via the property `hackairTBox:involvesRecommendation`), with the message "*You can still go on a picnic but pay attention for changes in air quality*" as a relevant advice from the hackAIR system. More details about the recommendation process are given in Section 6.

3.4.2 The hackAIR SPIN Rules

The basic functionality that the ontology rule layer aims to cover is the interpretation of information stored in lower levels of the ontology (TBox and ABox) in a uniform, automatic and semantically related way. In summary, interpretations in ontology-based frameworks have the following characteristics:

- they involve several concepts and their representations (individuals, relations, etc.);
- they describe data in different terms (additional classification, assertion to qualitative values, new values from rules that are met, etc.), by taking into account the context and conceptual knowledge represented on each concept and each individual in the ontology;
- they infer knowledge and facts that are not explicitly defined in the data but in the form of rules.

Thus, for the semantic interpretation and inference of new knowledge, we proceed with the implementation of a rule-based reasoning layer that fully adopts and implements the SPARQL Inferencing Notation (SPIN)[45] framework. SPIN is

[45] http://spinrdf.org/

a well-established standard to represent SPARQL rules and constraints on Semantic Web models [Knublauch et al., 2011]. Implemented SPIN rules run on top of both abstract ontology declarations (TBox) and actual assertions (ABox), which in turn serve as the basic knowledge (data, relations, semantics) fed into the rule-based framework for reasoning new assertions.

SPIN utilities are available for use by importing the SPIN vocabulary[46]. In the hackAIR ontology, we import/adopt and extend specific SPIN components (*SPIN rules*, *SPIN magic properties* and *SPIN functions*) in the *hackAIR SPIN Rules layer*. A detailed description about the specifications and implementation of SPIN rules as well as the use cases that the rules cover within the context of the hackAIR reasoning framework, is given in Section 6.3.

3.4.3 Mapping hackAIR ontology into existing concepts

In the current section we describe the results of integrating the hackAIR concepts with existing relevant concepts from external ontologies/vocabularies, presented in Section 3.1.3. We have concentrated in classes and properties that could be considered as similar, in terms of semantics, relations and the context where they are defined. The so called *ontology mapping* enables the establishment of semantic interoperability between new and existing sources, by defining the direct linking of classes/properties with third party ontologies and standards.

For the discrete definition of ontology mappings, we create a separate ontology document (presented in Figure 7) that contains the mappings between the hackAIR ontology concepts and those of third party vocabularies/ontologies. This document facilitates the direct alignment and easy comprehension of terms, data and relations from multiple domains. The linking between existing and new classes and properties is established via the declaration of `rdfs:subClassOf` and `rdfs:subPropertyOf` relations correspondingly. By linking concepts with the aforementioned relations, we inherit all involved semantics declared in external ontologies of the involved entities. For example, considering a declaration like the following:

```
<myOntology:ClassA rdfs:subClassOf externalOntlology:ClassB>
```

means that `myOntology:ClassA`, as being sub-class of `externalOntlology:ClassB`, is a more specialised concept that its superclass which is more general). Also, semantics and relations asserted in `externalOntlology:ClassB` can be also asserted in `myOntology:ClassA` and in all of its subclasses. This way, we achieve to not only adopt existing classes but to also extend them, with respect to the context of our domain.

Information of the ontology mapping document (Figure 7) is presented in TURTLE format and relations can be considered as valid at the date of publication of this document.

[46] http://spinrdf.org/spl

```
@prefix hackairMappings:    <http://mklab.iti.gr/hackair/hackair_mappings> .
@prefix hackairTBox:        <http://mklab.iti.gr/hackair/hackairTBox#> .
@prefix hackairABox:        <http://mklab.iti.gr/hackair/hackairABox#> .
@prefix owl:                <http://www.w3.org/2002/07/owl#> .
@prefix rdf:                <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:               <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dc:                 <http://purl.org/dc/terms/#> .
@prefix foaf:               <http://xmlns.com/foaf/0.1/> .
@prefix pescadoPDL:         <http://www.pescado-project.eu/ontology/pescadoPDL.owl#> .
@prefix pescadoData:        <http://www.pescado-project.eu/ontology/pescadoData.owl#> .
@prefix pescadoDiseases:    <http://www.pescado-project.eu/ontology/pescadoDiseases.owl#> .
@prefix dce:                <http://purl.org/dc/elements/1.1/> .
@prefix prov:               <http://www.w3.org/ns/prov#> .
@prefic sweet_indi:         <http://sweet.jpl.nasa.gov/2.3/propIndex.owl#> .
@prefic sweet_thermo:       <http://sweet.jpl.nasa.gov/2.3/propTemperature.owl#> .
@prefix sweet_aero:         <http://sweet.jpl.nasa.gov/2.3/matrAerosol.owl#> .
@prefix sweet_units:        <http://sweet.jpl.nasa.gov/2.3/reprSciUnits.owl#> .
@prefix geonames:           <http://www.geonames.org/ontology#> .
@prefix linkedGeoData:      <http://linkedgeodata.org/ontology/> .
@prefix odpObservation:     <http://www.ontologydesignpatterns.org/cp/owl/observation.owl#> .
@prefix odpAction:          <http://www.ontologydesignpatterns.org/cp/owl/Action.owl#> .
@prefix odpPlace:           <http://www.ontologydesignpatterns.org/cp/owl/place.owl#> .
@prefix odpActivityPattern: <http://descartes-core.org/ontologies/activities/1.0/ActivityPattern#> .

<http://mklab.iti.gr/hackair/hackair_mappings>
        rdf:type owl:Ontology ;
        dc:title "hackAIR Ontology Mappings to external classes/properties" ;
        dc:publisher "CERTH-ITI" ;
        dc:creator "Marina Riga" ;
        dc:description "This document contains the mappings of the hackAIR ontologies' terms to third
       party vocabularies for ease of alignment of terms and data from multiple domains." ;
        rdfs:comment "Those mappings are non-exhaustive and valid at the date of publication. They
       can be invalidated by further changes in the quoted ontologies." .

hackairTBox:Person
    rdfs:subClassOf foaf:Person, pescadoPDL:User, dce:Agent, prov:Person .

hackairTBox:Activity
    rdfs:subClassOf pescadoPDL:Activity, prov:Activity, odpAction:Action, odpActivityPattern:Activity .

hackairTBox:Location
    rdfs:subClassOf dce:Location, prov:Location, odpPlace:place .

hackairTBox:LocationCity
    rdfs:subClassOf geonames:city, linkedGeoData:City .

hackairTBox:LocationCountry
    rdfs:subClassOf geonames:country, linkedGeoData:Country .

hackairTBox:Request
    rdfs:subClassOf pescadoPDL:Request .

hackairTBox:HealthProblem
    rdfs:subClassOf pescadoDiseases:Disease .

hackairTBox:EnvironmentalData
    rdfs:subClassOf pescadoData:EnvironmentalData, odpObservation:Observation .

hackairTBox:Index
    rdfs:subClassOf pescadoData:IndexValue, sweet_indi:EnvironmentalIndex .

hackairTBox:Unit
    rdfs:subClassOf sweet_units:Unit .

hackairTBox:AirPollutantIndex
    rdfs:subClassOf sweet_indi:AirQualityIndex .

hackairTBox:ParticulateMatter
    rdfs:subClassOf sweet_aero:Particulate .

hackairTBox:PM10
    owl:sameAs sweet_aero:PM10 .

hackairTBox:PM2_5
    owl:sameAs sweet_aero:PM2point5 .

hackairTBox:hasName
    rdfs:subPropertyOf foaf:name .

hackairTBox:hasAge
    rdfs:subPropertyOf foaf:age .

hackairTBox:hasDescription
    rdfs:subPropertyOf dce:description .

hackairTBox:hasSubmissionDateTime
    rdfs:subPropertyOf dce:date .

hackairTBox:hasLocation
    owl:inverseOf odpPlace:isLocationOf .

hackairABox:degreesC
    rdf:type hackairTBox:Unit, sweet_thermo:Temperature .
```

Figure 7 – Ontology mapping between hackAIR and existing notions

3.5 A use case scenario

For demonstrating the hackAIR ontology and the way unstructured data is transformed into ontology-based structured data, we decided to follow a real scenario called *Personas scenario I*, as described in the *hackAIR user and technical requirements analysis* [hackAIR D2.2, 2016]. In the following paragraph, we focus in those parts of the scenario that are essential to be represented in the hackAIR knowledge base and recommendation system via relevant ontology notions:

> *"… **Karl (63)** is a retired teacher. Karl used to live in a highly polluted industrial area with **his daughter Anna** when she was a child. At the age of 10, **Anna was diagnosed with Asthma**. … When they **moved to Berlin**, Karl became a member of … Karl's daughter, **Anna, is a 32-year-old woman with asthma** and also **living in Berlin**. She lives only a few streets away from her father. Anna **is five months pregnant** of her first child. Since her pregnancy, Anna would like to get current local air quality information on a daily basis. … Anna has a busy job as an architect and has little spare time left to fully engage in this. Anna has moderate technical skills, a cheap Android smartphone and . … Anna is married to **Stephan**, a **35-year-old** graphic designer in a local communication agency. In his free time, Stephan **trains to participate in an occasional half marathon**. … He owns the latest iPhone and **goes to work by bike**. … **Karl has created a profile on the platform, and he has indicated he has asthma**. This way, when he gets an air quality notification on his tablet for people with asthma, **he can inform his daughter** on this. … **Anna doesn't like to install** new applications on her phone because it has very little memory space left."*

Table 11 presents how free text in natural language is converted into ontology triples, by following the schema described in hackAIR TBox ontology. Here, we focus only on information related to the user profile data.

Table 11 – Conversion of unstructured text into relevant hackAIR ontology notions, with respect to the user profile's details

Unstructured text	Triples (in TURTLE format[47])
About Karl	

Karl (63)	`hackairABox:Karl`
	`  rdf:type hackairTBox:Person ;`
	`  hackairTBox:hasAge 63^^xsd:integer ;`
	`  hackairTBox:hasGender hackairTBox:male ; .`

Berlin	`hackairABox:Berlin`
	`  rdf:type hackairTBox:LocationCity ;`
	`  hackairTBox:hasName "Berlin"^^xsd:string ; .`

moved to Berlin	`hackairABox:Karl hackairTBox:hasLocation hackairABox:Berlin .`
About Anna	

Anna, is a 32-year-old woman with asthma	`hackairABox:Anna`
	`  rdf:type hackairTBox:Person ;`
	`  hackairTBox:hasAge 32^^xsd:integer ;`

[47] This is a more compact format of representing triples than rdf/xml or n-triples; though, they do have the same representation base which is to mention statements in the form of subject-predicate-object.

	hackairTBox:hasGender hackairTBox:female ; hackairTBox:isSensitiveTo hackairTBox:Asthma ; .
living in Berlin	hackairABox:Anna hackairTBox:hasLocation hackairABox:Berlin .
is 5 months pregnant	hackairABox:Anna hackairTBox:isPregnant "true"^^xsd:boolean .
goes to work by bike or subway	hackairABox:Anna hackairTBox:availableMOT hackairTBox:bike ; hackairTBox:availableMOT hackairTBox:public_transport ; hackairTBox:hasPreferredActivity hackairTBox:biking_activity ; .
About Stephan	
Stephan, a 35-year-old	hackairABox:Stephan rdf:type hackairTBox:Person ; hackairTBox:hasAge 35^^xsd:integer ; . hackairTBox:hasGender hackairTBox:male ; .
trains to participate in an occasional half marathon	hackairABox:Stephan hackairTBox:hasPreferredActivity hackairTBox:jogging_activity .
goes to work by bike	hackairABox:Stephan hackairTBox:hasPreferredActivity hackairTBox:biking_activity .
Additional instantiations	
Karl creates involved profile for Anna	hackairABox:Karl hackairTBox:hasRelatedPerson hackairABox:Anna .

If we examine closer the information involved in the scenario, we see that Anna is not planning to be a direct user of applications such as the hackAIR platform; she does not want to install new apps in her mobile phone due to inadequate free memory space. On the other hand, Karl wants to keep her informed about current AQ condition in her area, so he is planning to create a profile with respect to her needs. Karl can create two hackAIR profiles that are interlinked, in order to distinguish his profile characteristics (elderly person) from her daughter's ones (pregnant, asthma, etc.). Both profiles will be populated in the ontology as individuals of `Person` type, as seen above, and additional triples will be created to interlink the profiles (<`hackairABox:Karl hackairTBox:hasRelatedPerson hackairABox:Anna`>). This way we achieve to handle both profiles with one single request for recommendation. The RS will take into account the characteristics of each profile separately and infer useful recommendations with respect to them.

Details on how information related to the requests for recommendation and the results from the reasoning process are presented in Sections 4.3 and 6.4 correspondingly.

3.6 Ontology metrics and evaluation

The evaluation of ontologies is an emerging field of research in the Ontological Engineering community that deals with the problem of assessing a given ontology from the point of view of a particular criterion of application. Existing ontology evaluation methods generally propose automated or semi-automated approaches that focus in specific qualitative (number of classes, properties, axioms, etc.) or quantitative criteria (consistency, completeness, expandability, sensitiveness, etc.) used to assess the examined ontology. An integrated review of ontology evaluation methods is attributed in publications [Gangemi et al., 2005] and [Brank et al., 2005]. Such techniques will help uncover errors in implementation, and inefficiencies regarding the modelling, the complexity and size of the ontologies. Nevertheless, no evaluation method (either as stand-alone or in combination) can guarantee a good ontology; on the contrary, it can definitely recognize problematic parts of it in terms of structure and consistency [Vrandečić, 2009].

For the current task, we perform metric-based ontology quality analysis to examine from a quantitative perspective the ontology quality. We adopt specific techniques and tools that evaluate both the consistency and the structure of the hackAIR ontology, regardless of the domain that they describe. The main approach and results derived are described in the following sub-sections.

3.6.1 Evaluating the structure

For evaluating the structure of the hackAIR ontology, we used OntoMetrics[48], a web-based tool that validates and displays statistics about specific parameters, like the following:

- **Base metrics[49]:** counters for classes, object/data properties, class/equivalent class/disjoint class axioms, domain/range declarations, individuals, etc.
- **Schema metrics[50]:** attribute/inheritance/relationship richness; and
- **Knowledgebase metrics[51]:** average population, class richness.

In the following table we present the results derived from the analysis of two specific layers in the hackAIR ontology: the Tbox (schema) and the SPIN layer (rules). The ABox (assertions) was excluded from metrics results since no structural data is included in the ontology but only instantiations referring to specific users, locations, observations and request.

Table 12 – Ontology metrics produced by OntoMetrics tool

	Metric name	TBox layer	SPIN layer
Base Metrics	Class count	125	138
	Object property count	31	31
	Object property domain axioms count	27	27
	Object property range axioms count	27	27

[48] Available at: https://ontometrics.informatik.uni-rostock.de
[49] Description of this type of metrics is available at: https://ontometrics.informatik.uni-rostock.de/wiki/index.php/Base_Metrics
[50] Description and calculation formulas available at: https://ontometrics.informatik.uni-rostock.de/wiki/index.php/Schema_Metrics
[51] Description and calculation formulas available at: https://ontometrics.informatik.uni-rostock.de/wiki/index.php/Knowledgebase_Metrics

	Data property count	14	14
	Data property domain axioms count	14	14
	Data property range axioms count	14	14
	Individual count	230	624
	Axioms count	1,106	14,447
	SubClassOf axioms count	124	114
	Equivalent classes axiom count	16	16
	Disjoint classes axiom count	75	75
	Class assertion axioms count	254	1,462
	Object property assertion axioms count	126	126
	Data property assertion axioms count	249	249
	DL expressivity	*ALCHOI(D)*	*AL*
Schema metrics	Attribute richness	0.112	0.0
	Inheritance richness	0.992	0.826087
	Relationship richness	0.495935	0.0
Knowledgebase metrics	Average population	1.84	1.355072
	Class richness	0.544	0.173913

As seen in Table 12, the hackAIR TBox ontology contains an adequate number of classes, and quite a number of object properties and datatype properties, used to make assertions on the individuals described in the ontology. SPIN imports the overall schema described in TBox and contains additional classes for the instantiation of SPIN rules; object and data properties are inherited from TBox. The analysed ontology layers contain a relevant number of individuals and assertions on them; these individuals are used in the schema to represent e.g. the types of environmental data, the levels of AQI, the units of measurement, attributes that characterise some ontological classes, etc., but the largest part is held by descriptions that carry messages and details of the defined tips of the day and personalised recommendations.

Both examined parts of hackAIR ontology are quire rich in terms of class restrictions used to characterise the terminological knowledge to be represented, as shown by the axiom counters in Base metrics part of Table 12. Rules defined in SPIN layer ontology, which are attached in relevant classes, are categorised by OntoMetrics as axioms. To conclude with analysed base metrics, the DL expressivity information tells in which Description Logics (DL) variant the

ontology can be categorised. The expressivity is encoded in labels[52]; in our case, the following DL characteristics were identified: (i) "*AL*" or "*ALC*" stands for the attributive (base) language which allows atomic negation, concept intersections, universal restrictions and existential quantification, (ii) "*H*" stands for the existence of role hierarchy (subproperties – rdfs:subPropertyOf), (iii) "*O*" stands for the use of nominals (enumerated classes of object value restrictions), (iv) "*I*" stands for the definition of inverse properties, and (v) "*(D)*" stands for the use of datatype properties, data values or data types.

Concerning schema metrics, quantitative values were estimated for expressing the attribute, inheritance and relationship richness (definitions explained in [Tartir et al., 2010]). More specifically, the attribute richness is the average number of attributes (object/datatype properties) per class. The number of attributes that are defined for each class can indicate the amount of information that is applicable, i.e. can be asserted, to instance data. This metric is not calculated for hackAIR SPIN layer since the ontology does not contain any new attributes declared.

Inheritance richness is defined as the average number of subclasses per class; it is a good indicator of how well knowledge is grouped into different categories and subcategories in the ontology. This measure can distinguish a horizontal ontology (where classes have a large number of direct subclasses) from a vertical ontology (where classes have a small number of direct subclasses). The examined hackAIR ontologies are of deep (vertical) type which indicated that the ontology covers a specific domain in a detailed manner.

Relationship richness is defined as the ratio of the number of non-inheritance relationships (i.e. object properties, equivalent classes, disjoint classes) divided by the total number of inheritance (i.e. subclass relations) and non-inheritance relationships defined in the ontology. This metric reflects the diversity of the types of relations in the ontology. These relations are contained in both the TBox and the SPIN layer, but the tool seems not to be able to identify them in the SPIN rules layer.

Finally, concerning knowledgebase metrics, TBox has an adequate class richness result, meaning that instances in the ontology are quite well distributed across classes; the ratio is smaller in SPIN layer, meaning that included data do not exemplify all the knowledge in the schema. The average population metric results has relevant interpretation, as being an indicator of the number of instances compared to the number of classes defined in the ontology.

3.6.2 Evaluating the consistency and quality

For the task of evaluating the consistency and overall quality of the hackAIR ontology we used the software named OOPS (OntOlogy Pitfall Scanner), which is a web tool for detecting the most common pitfalls[53] in ontologies [Poveda-Villalón et al., 2009]. The results given by OOPS suggest how the ontology elements could be modified in order to improve the ontology quality; these suggestions should be interpreted manually so as to be revised properly. In the evaluation process we included only the schema (hackAIR TBox ontology) since the assertions (hackAIR ABox ontology) follow the same structure. The analysis of the hackAIR SPIN ontology was not applicable with this tool, since OOPS couldn't conceive properly the rules expressed in SPIN standard.

The system provides an indicator for each pitfall, according to their possible negative consequences:

- critical pitfalls, the correction of which are crucial and will affect the ontology consistency and reasoning;
- important pitfalls, which are not critical for ontology functioning, but are considered as important to be corrected; and
- minor pitfalls, which do not cause any actual problem but by correcting them will make the ontology clearer and more compact.

[52] More details available at: https://en.wikipedia.org/wiki/Description_logic#Naming_convention
[53] A catalogue of common pitfalls is given in http://oops.linkeddata.es/catalogue.jsp

In Table 13 we present the pitfalls appeared while evaluating our schema, along with a brief description of their meaning and together with the number of cases for which they were specified.

Table 13 – Ontology's pitfalls detected by OOPS.

No.	Pitfall description	Results
#1	Missing annotations (*Minor*) Ontology terms lack annotation properties that would improve the ontology understanding and usability from a user point of view.	169 cases
#2	Creating synonyms as classes (*Minor*) Classes whose identifiers are synonyms, could be defined as equivalent (`owl:equivalentClass`) in the same namespace.	2 cases
#3	Missing domain or range in properties (*Important*) Relationships and/or attributes without domain or range are included in the ontology.	7 cases
#4	Inverse relationships not explicitly declared (*Minor*) A relationship has no inverse relationship defined.	27 cases
#5	Using different naming conventions in the ontology (*Minor*) The ontology elements are not named following the same convention.	ontology

According to the OOPS tools, one **important** pitfall exists (see pitfall #3 in Table 13) in the hackAIR ontology that is the absence of domain and range declarations for 7 different cases. We solved this pitfall by carefully examining the derived object properties that had the aforementioned problem and we explicitly defined the relevant domain and range declarations in the ontology that were missing. The rest of the recognised pitfalls were categorized as **minor** and for handling them, specific actions were taken as described next.

For pitfall #1 there were 169 cases spotted by the tool where annotations and descriptions were missing in our ontology; to overcome this pitfall and to improve comprehension and expressiveness of the ontology, we assigned human readable annotations to every defined concept in the ontology, with the adoption of properties `rdfs:label`, `rdfs:comment` or `dc:decription`.

Concerning pitfall #2, we didn't make any alteration in the ontology; the tool wrongly pointed out that two of the declared classes should be considered as the same concept; these classes were the `hackairTBox:SportsWalkingActivity` and the `hackairTBox:SportsWalkingActivity`, which obviously represent different concepts in our domain.

Regarding pitfall #4, the missing declarations of inverse relationships were restored properly in order to improve the completeness of the ontology. Pitfall #5 arose due to the fact that some notions in our ontology (usually the instances of classes) have different naming convention that declared classes and properties. Since this pitfall does not affect the structure or comprehensibility of the ontology, there was no need to take any action.

4 Problem Description Language (PDL)

A Problem Description Language (PDL) is an integral part of a decision support system, responsible for the representation of a problem (*request*) in a formal and comprehensible way. In the hackAIR system, the PDL should act as an intermediate module for the communication between the hackAIR UI and the DS module. The hackAIR PDL aims to specify the language and structure that is used indirectly by humans and directly by the system, in order to formulate a request regarding environmental aspects which is intended to be served by the hackAIR system.

According to the design of the hackAIR KB and DSS architecture, the decision support to the user will be implemented by reasoning techniques and strategies based on the information stored in the ontologies; the schema, the assertions and the rules composing the hackAIR ontologies should be consistently orchestrated with the principles, the notions and the information formulated by the PDL. For that reason we selected to structure the hackAIR PDL on OWL Web Ontology Language, the notions and the schema of which will be part of the hackAIR TBox ontology.

In the following sub-sections, we present the proposed hackAIR PDL, giving in detail the structure and details on how the request of the user is formulated so as to be properly submitted for decision support by the hackAIR RS. We describe the requirements that were taken into account in the design process of the PDL, the main entities of the PDL and we conclude with example PDL representations in real use case.

4.1 User needs with respect to decision support

Within the hackAIR context, every request for decision support is inextricably linked to specific characteristics and details of the user profile; the so called personalised (i.e. user-profile driven) recommendations should be the main functionality of the hackAIR RS (details in Section 6) and recommendations will be relevant to the needs of specific user profiles (details in Section 6.1.2). We ended up on these features, after: (i) having examined relevant decision support services available in the market that target in the environmental domain, (ii) a thorough analysis from experts of the consortiums as well as external ones, regarding the types/characteristics of vulnerable individuals to hazardous AQ conditions, and (iii) performing co-creation workshops [hackAIR D2.4, 2017] with potential users of the system. It was found out that personalised recommendations that are especially targeted to *preferred outdoor activities* and specific *health sensitivities* can be perceived as something motivating, or else of great impact, that would trigger them to use the hackAIR platform in order to be timely and accurately informed and to promote a behavioural change for the benefit of their health.

Typical decision support scenarios, with respect to the aforementioned characteristics and thus within the context of the hackAIR personalised recommendation process, could be the following:

- *"I have **asthma** and I am planning to go for a **long walk** to the **city centre** today. Should I postpone my activity?"*
- *"I am a **teacher** and I want to be **pro-active** for my **children** at school, in case of bad air quality conditions in this area."*
- *"My father who is an **elder person** with **health sensitivities** wants to go **for picnic** in the city park. What does the system recommend with respect to his health condition and to the existing air pollution levels?"*

In the above requests for recommendation, expressed in natural language text, we highlight in bold font those concepts that are significant and need to be covered by notions in the hackAIR ontology, for decision support by the hackAIR RS. Those use cases and concepts are primarily handled by relevant hackAIR PDL entities, as described next.

4.2 Definition of the hackAIR PDL

The specification of the first version of the hackAIR PDL is based on OWL, the established web ontology language which serves as the technological foundation for both the hackAIR KB and RS; the representation method of the PDL conforms

to the overall ontology-based approach made in hackAIR for semantic integration and reasoning purposes handled by the hackAIR system.

In examples presented in Section 4.1, there are common concepts which are typically referred within the context of user requests for decision supports; these are:

- information related to the user(s) or their related user profile;
- details about the area of interest, for which the user requests for environmental related information or recommendations;
- information of the kind of activity (or activities) the user wants to perform.

In order to support the recommendation process, an additional type of information is needed: the actual observation value of monitored environmental data in the area of interest. This information is provided by the *Data Fusion M*odule and should be supplementary included in the ontology representation of the request.

All the above specify the main building blocks for representing the problem description as defined in the hackAIR PDL. Each of the components of the PDL is described below.

4.2.1 Main entities

The problem of formulating a user request within the hackAIR context, involves the following notions of the hackAIR PDL: `Request`, `Person`, `Location`, `EnvironmentalData`, `Activity` and `Recommendation`. The interlinking between entities is feasible via corresponding properties as depicted in Figure 8.

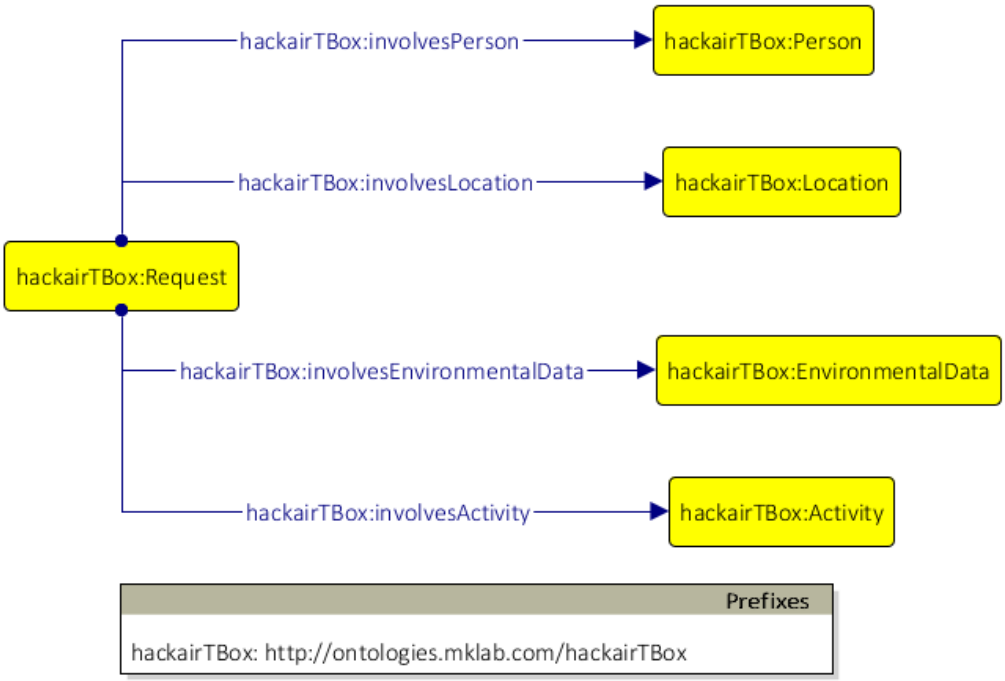


Figure 8 – Main entities involved in the hackAIR PDL

The generic concept that describes the problem is the class `Request`, details of which are already presented in Section 3.4.1. The properties that can be asserted in an instance of `Request` type are:

- `involvesPerson`: relates a specific request that arrives to the system with the user that performs the request for recommendation. The instance of class `Person` contains all user-profile related information that

determines the final result for decision support. Information about the user(s) submitting a request is clearly relevant to the capability of providing an adequate recommendation.

- `involvesLocation`: relates a specific request to the area of interest, for which environmental data should be available by the Data Fusion Module.
- `involvesEnvironmentalData`: relates the instance of `Request` with the current air quality conditions reported by the Data Fusion Module for the specific area of request.
- `involvesActivity`: relates the specific request to the activity or activities that the user wants to perform; all activities supported by the hackAIR ontologies can be utilised here. Usually activities linked to a specific request are those that are defined as preferred activities in the involved user profile. It should be noted that assertion of this type is optional - there can be requests for decision support that do not include any specific type of activity (e.g. a request for recommendation for an elderly person with no preferred activity defined in the system, is still valid and processable.

4.2.2 User – PDL interaction

As already stated, the PDL is a formal description of a user request for decision support by the hackAIR RS. The PDL interferes between the hackAIR UI and the KB and RS modules. There is no direct interaction between the PDL and the user, thus no demand for the user to express his/her requests in the aforementioned ontology-based format. The syntax and complexity of the PDL is hidden under specific web forms of the hackAIR platform, provided for adjusting the different profile characteristics by the user.

All information that is provided by the user via the hackAIR interface, together with the air quality data coming from the Data Fusion Module, are inserted into the KB, with the adoption of the PDL formalisation guidelines. The PDL facilitates a one-to-one mapping between sourced information (profile characteristics, AQ observation, requests, etc.) and data instantiated in the hackAIR ontology. This process enables the proper structuring of data for the rule-based reasoning system to perform user-profile driven decision support efficiently.

Details on the communication between the hackAIR UI and the ontology-based hackAIR modules (data input, post-process based on PDL utilities, data output) will be given in Section 5, while an abstract description of the step-by-step process of PDL-based information population follows next.

4.2.3 A step-by-step-process

Considering that all information relevant to a request is available, the following steps should be followed in order to formulate the problem of the user (request), according to PDL and ontology specifications:

1. An individual belonging to the `Request` is defined in the hackAIR ABox ontology, to uniquely identify the current problem under formalisation.
2. A linking between the individual of `Request` type and the individual of the involved person is instantiated via the property `involvesPerson`. If the user is already populated in the hackAIR ABox, the individual of `Person` type is recalled and used in that `Request-Person` relation. Either wise, an individual of class `Person` is populated, as demonstrated in Section 3.5, with instantiations of user-profile related information (such as *age*, *health sensitivities*, *preferred activities*, *location* of interest, other *related profiles*), wherever this is feasible.
3. A linking between the individual of `Request` type and the individual of the involved location of interest is instantiated via the property `involvesLocation`. If the location is already populated in the hackAIR ABox, the individual of `Location` type is recalled and used in that `Request-Location` relation. Either wise, an

individual of class `Location` is populated, as demonstrated in Section 3.5, with instantiations of location related information (such as *name of city* or *country*), wherever this is feasible.

4. An individual of `EnvironmentalData` type is populated, to represent information relevant to the current air quality condition for the location of interest. A linking between the individual of `Request` type and of newly populated `EnvironmentalData` individual is created via the property `involvesEnvironmentalData`.
5. A linking between the individual of `Request` type and the individual(s) of the activity (-ies) considered as relevant to the involved user profile is/are instantiated via the property `involvesActivity` .This step is optional, because in any case data can be directly retrieved from the involved user profile details (see declaration of `hasPreferredActivity` property) via relevant SPIN rules; so before running any inference, no linking between request and involved activity(-ies) is provided.

4.3 A use case scenario described with hackAIR PDL

We present here a use case scenario where the problem is described in natural language, and we demonstrate how the contained information is formalised with the adoption of the hackAIR PDL. For demonstration purposes and for maintaining a continuation in the utilised content, we follow the *Personas scenario I* introduced in [hackAIR D2.2, 2016], which was also was referenced in Section 3.5, for demonstrating the population process of user-profile related data in the ontology.

As a reminder of the scenario details, we have three persons: *Karl* (elderly person, father of Anna), *Anna* (pregnant woman with health sensitivities, goes to work by bike, and has no interest in installing AQ-related apps; wife of Stephan) and *Stephan* (exercising intensively and goes to work by bike); all of them live in Berlin. Since it is mandatory to have observation data of the area of interest, for the PDL and the recommendation process, we consider deriving from the Data Fusion Module that the current AQ in Berlin for PM AOD is 1.2.

Table 14 presents how the above information is converted into ontology triples, by following the schema represented in the hackAIR TBox ontology. We focus on data related to the problem description (request) for decision support.

Table 14 – Conversion of unstructured text into relevant hackAIR ontology notions, with respect to the problem description (request) of the user for recommendation

Triples (in TURTLE format)
Request from Karl (including Anna)

`hackairABox:request_from_Karl`
`  rdf:type hackairTBox:Request ;`
`  hackairTBox:involvesEnvironmentalData hackairABox:AODEnvData_for_Berlin ;`
`  hackairTBox:involvesLocation hackairABox:Berlin ;`
`  hackairTBox:involvesPerson hackairABox:Karl ;`
`  hackairTBox:involvesPerson hackairABox:Anna ; .`

Request from Stephan

`hackairABox:request_from_Stephan`
`  rdf:type hackairTBox:Request ;`
`  hackairTBox:involvesEnvironmentalData hackairABox:AODEnvData_for_Berlin ;`
`  hackairTBox:involvesLocation hackairABox:Berlin ;`

```
  hackairTBox:involvesPerson hackairABox:Stephan ; .
```

Additional instantiations

```
hackairABox:AODEnvData_for_Berlin
  rdf:type hackairTBox:AODEnvironmentalData ;
  hackairTBox:hasEnvironmentalDataType hackairTBox:PM_AOD ;
  hackairTBox:hasNumericalValue hackairABox:AODValue_for_Berlin ; .
```

```
hackairABox:AODValue_for_Berlin
  rdf:type hackairTBox:AODValue ;
  hackairTBox:hasValueValue "1.2"^^xsd:double ; .
```

In the above instantiations, no direct declaration of the `involvesActivity` property is asserted to individuals of `Request` type; such information is automatically inferred data of SPIN rules, as explained in Section 6.3. However, all other relevant properties that link a request with the involved person, location and observation are mandatory in the population process of individuals of `Request` type, and otherwise the recommendation process cannot be supported efficiently due to missing data.

5 Dynamic ontology population

In the following sub-sections we analyse the problem of ontology population and present the most established approaches found in literature. We focus on the requirements for dynamic population process of involved data in the hackAIR knowledge base, and we justify our proposed approach according to the defined needs. We present in detail the architecture, as well as the methods and tools adopted for the implementation of the hackAIR web-service and we conclude with some example results of the dynamic population process in real use cases.

5.1 State of the Art

Ontology population is the process of augmenting the ontology with instances of concepts and properties and is part of the **ontology learning** process, which refers to the automatic or semi-automatic construction, enrichment and adaptation of ontologies [Maedche and Staab, 2001]. Ontology population does not change the structure of an ontology itself but only its set of concepts and relations, and can be performed either manually or (semi)automatically via the use of respective software tools. The latter case is mostly appropriate when the number of instances to be inserted into the ontology is significantly high and a fully manual population process in this context would be extremely tedious and time-consuming.

State of the art ontology population approaches are mostly addressed to retrieving instances from textual corpora (i.e. natural language text, like e.g. product catalogues) and mainly involve machine learning, text mining and natural language processing (NLP) techniques; indicative approaches are presented in [Buitelaar and Cimiano, 2008] and [Petasis et al., 2011]. Another stream of ontology population research attempts to retrieve instances from structured content, like e.g. spreadsheets [Han et al., 2008], XML files [Modica et al., 2001] and, more recently, Linked Data sources [Mitzias et al., 2016].

In this deliverable we focus on a very specific type of ontology population which we call **dynamic ontology population** and involves an ontology model being populated **at run-time**, contrary to the case of having a mostly static ontology that has been populated before the application was executed (e.g. during the ontology construction phase). Although not many relevant approaches currently exist in literature, this will most probably change due to the rapidly increasing use of ontologies in modern applications. For instance, the authors in [Niepert et al., 2008] propose the deployment of answer set programming to carefully-solicited expert feedback for dynamically populating and partially inferring an ontology. The paper focused on an appealing use case, the Stanford Encyclopedia of Philosophy (SEP), which is substantially complex and highly dynamic.

Two other applied approaches include the UIMAST dynamic ontology population tool [Fiorelli et al., 2010], which is an output from the UIMAST project[54], and the Magpie suite of semantic web tools[55]. In particular, Magpie's dynamic population focus, i.e. "*ontologies can be populated so that they reflect personalized constraints on a generic KB*" [Dzbor and Motta, 2006], strongly coincides with our strategic aim to dynamically populate the ontology with personalised information, in order to develop a platform that is capable of addressing personalised decision support.

Concerning the task of dynamic population of the hackAIR KB with heterogeneous data (user profile and environmental fused data), the need that arises is the communication of involved hackAIR modules in a direct, fast straightforward and interoperable way. By exploiting the nature and infrastructure of the hackAIR platform (mobile/web application), the required communication between the *data providers* (hackAIR User profile and Fused data modules) and the *data receivers* (hackAIR KB) can be accomplished with the development of relevant **web-services**; a web-service is any piece

[54] UIMAST project: http://semanticturkey.uniroma2.it/extensions/uimast/
[55] Magpie homepage: http://projects.kmi.open.ac.uk/magpie/main.html

of software that makes itself available over the internet and uses a standardised messaging system for exchanging data between applications, modules, or even different systems.

For the exchange of information and the communication between involved modules, the need is to adopt a lightweight and concise **data exchange language** that is domain-independent, produces small data size and supports fast transmission speeds. It is important to keep hackAIR services as lightweight and efficient as possible because, usually mobile devices are bound by a monthly data cap or operate with slow connectivity speeds, both of which facts do directly affect the response time and efficiency of the application. Among the most popular data formats (JSON[56], XML[57], REBOL[58], YAML[59], etc.) for web-based data exchange, JSON format is preferable and widely accepted due to its simple, flexible and compact way of representing data in a structured and uniform way, covering thus the aforementioned needs for our designed web-service.

Finally, for the process of mapping input (formatted) and targeted (ontology-based) data, an automated alignment of identical concepts will be performed, before the dynamic population process. This automatic process will hide the expert's knowledge behind its execution, meaning that the initial alignment between entities of input data and classes/relation in the ontology will be designed and implemented in the context of the web-service's functionality manually by the ontology expert. In our case, we are dealing with a small set of data with defined semantics, focused to the needs of the context of the current task. Any additional efforts for automating the process with machine learning and natural language processing techniques would be costly in terms of computational efficiency, transmission speed of data and response time of the application. In case the scale of the problem was bigger (multiple input data with diverse ontology concepts involved the transformation process) a more sophisticated approach would be examined.

5.2 hackAIR web-service for ontology population

For the process of dynamic ontology population, we selected to implement a RESTful API. The Representational state transfer (REST) is the style of software architecture that basically exploits the existing technology and protocols of the web [Fielding and Richard, 2002]. Thus, RESTful web services are built to work on the Web for establishing communication between different modules. Here, the need is to create a service that the hackAIR UI will be able to call, in order to pass data derived from the *User profile module* and *the Data Fusion Module* into the hackAIR *ontology-based KB*. The ontology population process will be triggered and served dynamically, upon user request (*http request*) to the hackAIR system for providing decision support inferences.

The hackAIR API should demonstrate the following functionality (Figure 9):

- it takes as input textual data in a pre-specified format; these data will be instantiated in the ontology;
- it handles each individual statement in input data, and maps their content into corresponding ontology notions and values; as a result, it transforms the initial statements ontology statements (triples);
- the newly created triples are populated as new assertions in the existing ontology.

The service has been implemented in Java (Java EE 7 SDK[60]), with the adoption of additional frameworks:

- **Apache Jena**[61]: a free and open source Java framework for building Semantic Web and Linked Data applications. It includes the RDF API that facilitates the creation and manipulation of RDF graphs. It enables

[56] http://www.json.org/
[57] https://www.w3.org/XML/
[58] http://www.rebol.com/
[59] http://yaml.org/
[60] https://docs.oracle.com/javaee/7/index.html
[61] https://jena.apache.org/

also the serialization of the schema in any of the most popular ontology formats, such as RDF/XML, RDF/OWL, TURTLE, etc.

- javax.ws.rs library[62]: a built-in package included in the Java Enterprise Edition (EE) distribution for the development of RESTful web-services.
- GlassFish Server[63]: an open-source application server produced by Sun Microsystems for the Java EE platform, utilised for handling HTTP queries to the RESTful API.
- json-simple[64]: a well-known java toolkit for parsing (encoding/decoding) JSON text.

In Figure 10 we present the list of dependencies as configured in our Java project and handled by the Maven project management tool[65].

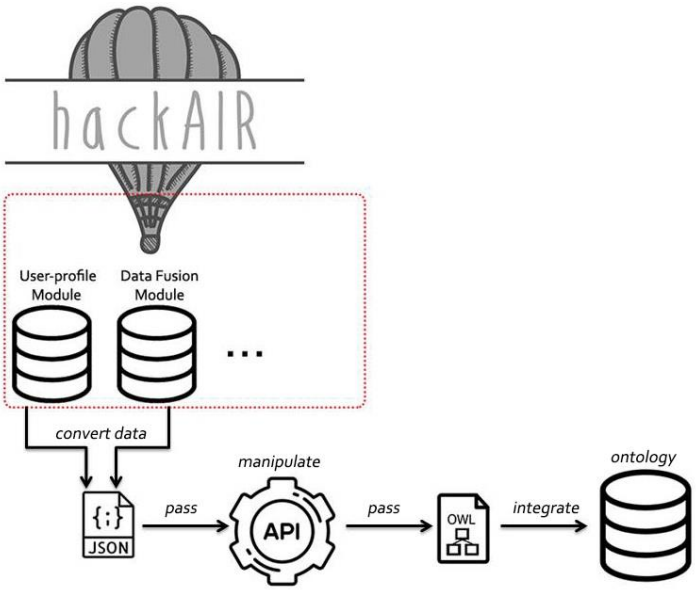


Figure 9 – The input and output data of the RESTful API

[62] http://docs.oracle.com/javaee/6/api/javax/ws/rs/package-summary.html
[63] http://www.oracle.com/technetwork/middleware/glassfish/overview/index.html
[64] https://github.com/fangyidong/json-simple
[65] https://maven.apache.org/

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>iti.mriga</groupId>
    <artifactId>hackAIR_project</artifactId>
    <version>1.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>hackAIR_project</name>

    <properties>
        <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.jena</groupId>
            <artifactId>jena-core</artifactId>
            <version>3.1.0</version>
            <type>jar</type>
        </dependency>
        <dependency>
            <groupId>javax</groupId>
            <artifactId>javaee-web-api</artifactId>
            <version>7.0</version>
            <scope>provided</scope>
        </dependency>

        <dependency>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1</version>
        </dependency>
    </dependencies>
```

Figure 10 – An excerpt of pom.xml document where project dependencies to external libraries are stated

5.2.1 Details of the hackAIR API

Here, we describe in detail the requirements and characteristics of the hackAIR web service developed within the scope of Task T4.2, for the orchestrations and dynamic population of user profile and fused data in the ontology. Currently, the API is not publicly available but runs as a local service; the public URI will be available for integrating the web-service with the 1st version of the hackAIR platform in M20 of the project (August, 2017). The API will respond under the URI:

<http://{baseURI}/hackAIR_api/dynamicPopulation>

For requests to the API we use the POST method, which is a request method supported by the HTTP protocol for submitting data to be processed by the service. The POST request method requests that the web server accept the data enclosed in the body of the request message, mostly for storing or further manipulation. POST method transfers information via HTTP headers, without any restriction on data size to be sent. An excerpt of the Java code implemented for handling the POST request can be seen in Figure 11.

```
@POST
@Path("/dynamicPopulation")
@Consumes("application/json")
public Response dynamicPopulationOfUserData(String userJSON) throws ParseException {
    JSONObject main_JSON = new JSONObject();

    try {
        JSONObject userJSONObject = (JSONObject) new JSONParser().parse(userJSON);

        UserProfile userProfile = UserProfile.createUserWithRelatedProfilesFromJSONobject(userJSONObject);

        /* Handle ontology related processes */
        MyModel myModel = new MyModel(hackAIRSPIN_URI_ttl);
        // Enrich ontology with user profile
        myModel.enrichOntologyWithFullUserProfile(userProfile);
    }
    catch (ParseException ex) {
        main_JSON.put("error", "JSON syntax error");
        return Response.status(Response.Status.BAD_REQUEST).entity(main_JSON.toString()).build();
    }

    main_JSON.put("ok", "Data are sucessfully populated in the ontology");

    return Response.status(Response.Status.CREATED).entity(main_JSON.toString()).build();

}
```

Figure 11 – An excerpt of Java code from the RESTful API for dynamic ontology population

As mentioned previously, the exchange of information between the involved components of the hackAIR platform will be based on JSON (JavaScript Object Notation), a text-based data interchange format that represents its content as simple key-value pairs, giving emphasis in both the content and the structuring of data. Thus, for describing the content of the request in JSON language, the following entities may be used:

- **Objects** – an object is an unordered set of name/pair values, bounded within left and right brace `{ }`. Each name is followed by colon `:` and the name/value pairs are separated by comma `,`;
- **Arrays** – an array is an ordered collection of values, bounded within left and right brackets `[ ]`. Values are separated by comma `,`;
- **Values** – a value can be a string in double quotes `""`, or a `number`, or `true`/`false` value, or even another `object` or an `array`.

On the basis of the aforementioned entities, we specify the mandatory and optional parameters that are accepted in the POST request of the developed hackAIR API, as summarised in Table 15:

Table 15 – JSON parameters of the POST request body

Parameter name	Parameter type	Mandatory / Optional	Accepted values
`username`	object	M	any `string` value

`gender`	object	M	**One of the following**:
			`male`
			`female`
			`other`

age	object	M	any integer value
locationCity	object	M	any string value
locationCountry	object	M	any string value
meansOfTransport	array	O	One of the following: bike bus car motorcycle train tram publicTransport
isPregnant	object	O	any boolean value
isSensitiveTo	object	O	One of the following: Asthma Allergy Cardiovascular Circulatory GeneralHealthProblem
isOutdoorJobUser	object	O	any boolean value
preferredActivities	object	O	preferredOutdoorActivities
preferredOutdoorActivities	array	O	One of the following: biking jogging motorcycling tennis swimmingOutdoors picnic playingInPark walking working generalActivity
airPollutant	object	M	Both: airPollutantName airPollutantValue

`airPollutantName`	object	M	One of the following:
			`PM_AOD`
			`PM10`
			`PM2_5`

`airPollutantValue`	object	M	any `double` value

For utilising JSON input in the dynamic population process, we have developed relevant Java classes and methods that get the formatted input and map all available name/value pairs into corresponding classes/properties assertions in the ontology; new triples are added in the hackAIR ABox for further processing by the RS. The POST request returns either an error message, if something goes wrong with the format of the JSON text, or a successful message that the population process was completed.

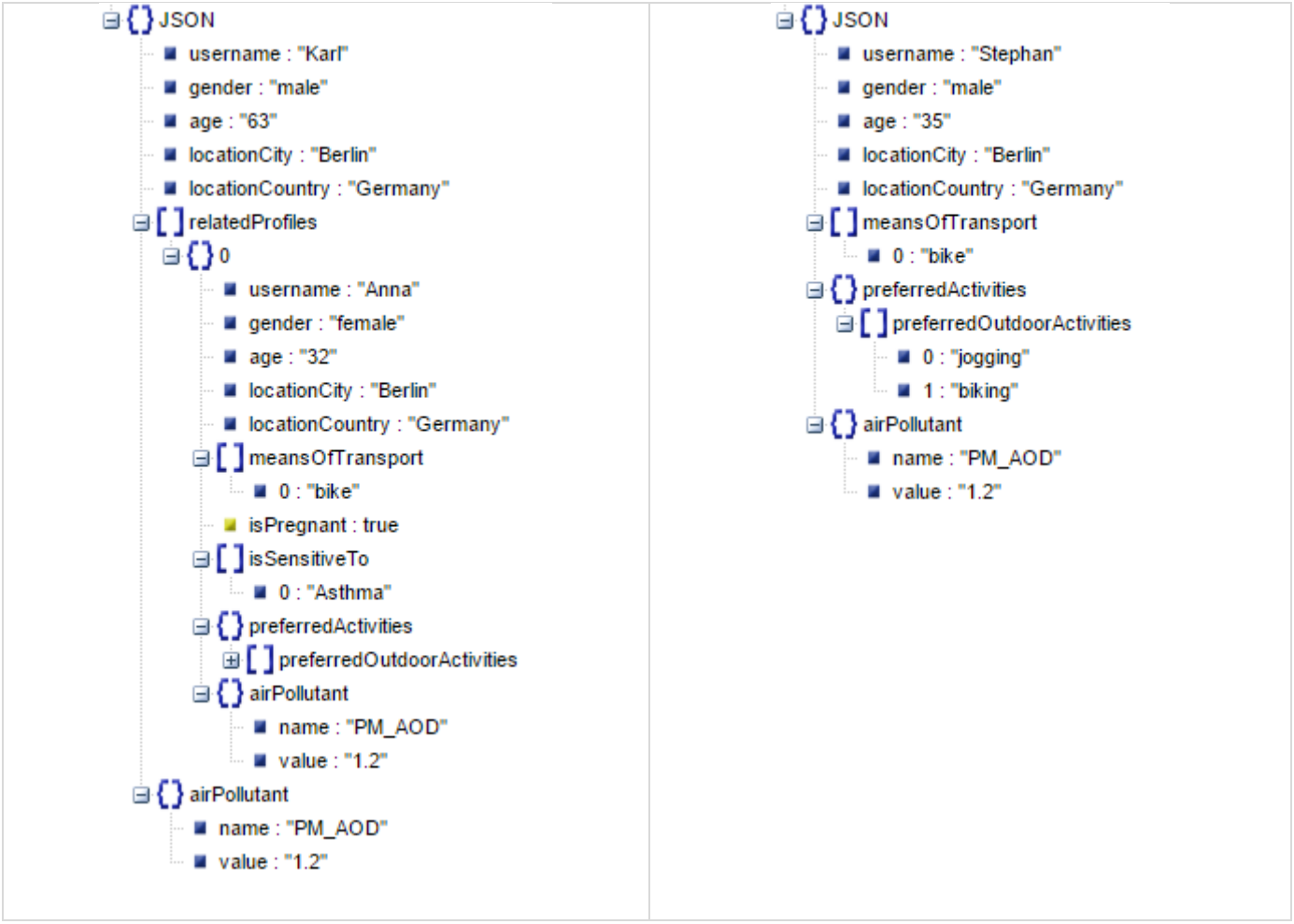
5.2.2 Example request to the hackAIR API

Considering the same use case scenario, as this was already presented in Sections 3.5 and 4.2.3 for demonstration purposes of the developed utilities, we present here the JSON inputs as they are formatted for two user scenarios:

- Karl, who is an elder person interested in getting recommendations for him and his daughter Anna, a 32-year old woman who is pregnant and has asthma.
- Stephan, a physically active person who likes running; he is married to Anna, and he is also interested for air quality related information.

Both JSON inputs include details about the different user profiles (pushed from the User Profile module) and existing AQ conditions (pushed from the Data Fusion Module). Visualisations in Figure 12 are produced with the use of the Online JSON Viewer[66]:

[66] Online JSON Viewer available at: http://jsonviewer.stack.hu/



(a) (b)

Figure 12 – JSON details of specific requests from: (a) Karl and his related profile (Anna), and (b) Stephan

In Figure 13 we present both POST queries as requested to the server for dynamic population. Both request resulted in ***HTTP Status: 201 Created*** response by the server, meaning that the request has been fulfilled and resulted in new created resources. The response time was approximately 850ms. Newly asserted entities that were populated in the ontology, were already presented in Section 4.3, where we examined the same use cases for forming user requests with the use of the hackAIR PDL.

```
POST /hackAIR_project/api/dynamicPopulation HTTP/1.1
Host: localhost:8080
Content-Type: application/json
{
    "username": "Karl",
    "gender":"male",
    "age":"63",
    "locationCity": "Berlin",
    "locationCountry": "Germany",
    "relatedProfiles": [{
            "username": "Anna",
            "gender":"female",
            "age":"32",
            "locationCity": "Berlin",
            "locationCountry": "Germany",
            "meansOfTransport": ["bike"],
            "isPregnant": true,
            "isSensitiveTo": ["Asthma"],
            "preferredActivities": {
                "preferredOutdoorActivities": ["biking"]
            },
            "airPollutant": {
                "name": "PM_AOD",
                "value": "1.2",
            }
        }],
    "airPollutant": {
        "name": "PM_AOD",
        "value": "1.2",
    }
}
```

Figure 13 – POST request as arrived to the GlassFish server

6 Rule-based reasoning framework

In this section, we deal with the specification of the reasoning techniques that sketch the common framework proposed for the realisation of a personalised recommendation module for providing decision support to the users of the hackAIR system; the implementation of the recommendation module is planned to be achieved during T5.2 "Component development and integration" of WP5 "Development of the hackAIR platform". The basic aim of the hackAIR reasoning module is to handle user-profile and environmental related data, analyse the semantics behind such content and infer recommendation(s) to the user, by selecting automatically the relevant ones from a list of pre-defined messages.

In the hackAIR, we select to provide to the users two types of recommendations: (a) *tips of the day* (see details in sub-section 6.1.1), which are useful daily advice on how to improve the ambient AQ and to reduce individual sources of air pollution, regardless of the user-profile content or of existing environmental conditions; (b) *personalised recommendations* (see details in sub-section 6.1.2), which can be conceived as pro-active measures (upon user request) on how to protect oneself from hazardous health effects of air pollution, targeting to specific characteristics of the user profile, like for example age, health status, preferred activities, etc.; users will be able to indicate certain options on their profile such as "I am pregnant", or "I have health sensitivities", or "I am an outdoor job worker", or even their everyday activities (e.g. running, walking, biking, etc.) and next, the recommendation system should take into account all the aforementioned factors, together with the existing environmental conditions for the location of the user.

Specific messages with fixed content were defined by hackAIR environmental experts for both types of recommendation, together with details on: (i) *how* and *when* recommendation messages will be provided, and (ii) *which type of user* such messages concern. The text of these messages will be instantiated in the ontology as individuals of `Recommendation` class, accompanied with information relevant to the aforementioned details: regarding the first aspect (how/when), tips of the day will be selected randomly to be presented to the user, while personalised recommendation will be provided upon request for decision support. Regarding the second aspect (which type of users), tips of the day could be provided to every direct user of the hackAIR system, while personalised recommendations only to valid types, i.e. a user with health sensitivities and with a specific preferred outdoor activity will get different recommendation from a health person that has the same preferred outdoor activity. Instantiations of personalised recommendations will carry such information that relates the message with the types of user for which the recommendation is valid.

Therefore, the problem of providing personalised inferences is transformed in a classification task where different recommendations are given to the users according to a specific set of classes in the ontology that they belong; for example, if a user belongs to *Class_A* and *Class_B* and the current AQ level has value *AQ_xyz*, then *recommendation_X* will be provided, or if a user belongs to Class_A, *Class_B* and *Class_C* and the same AQ condition exists, then *recommendation_Y* and *recommendation_Z* will be provided. The fixed types of users where an individual may belong, the strict declaration of involved semantics that describe their type, together with details describing each valid case per recommendation, do not leave any vague notions to affect the efficiency of the recommendation process. The absence of any type of uncertainty in information involved in the recommendation process rejects any uncertainty reasoning approach, as being non-applicable to the specified needs (Section 6.2.4).

Considering the context of requirements and functionality of the hackAIR recommendation system, we follow a hybrid reasoning approach with rules and ontologies, where an individual reasoner (Section 6.2.1) ensures the consistency of declared relations (hierarchy, domain, range, etc.), while for complex reasoning tasks we implement ontology rules, formulated according to the established SPIN standard (Section 6.3). The developed rules handle the data, semantics and relations represented in the ontology-based knowledge base and infer new relations between involved entities, to finally match personalised recommendations according to user needs and to existing AQ conditions.

In the following sub-sections we present the following: in Section 6.1 we give detailed specifications of the recommendation mechanism and utilities as defined within the context of the hackAIR project. We analyse the different use cases that the recommendation process should support and we give details about the set of user profiles and of the process of classifying a user into one or more relevant user classes (Section 6.1.2). In Section 6.2 we discuss the main characteristics of existing reasoning techniques and we justify our decision for SPIN rule-based implementation of the reasoning process. In Section 6.3 we continue with the analysis of the multi-layered reasoning approach, together with a detailed reference of example rules developed for the aforementioned task. We conclude in Section 6.4 with a real case scenario of requesting/providing recommendations, and we technically evaluate the results of the implemented rule-based reasoning framework in Section 6.5.

6.1 hackAIR recommendation requirements

The idea behind the hackAIR RS is to provide recommendations to the users in order to better inform citizens about (a) general facts/guidelines that lead to proven reduction of the individuals' emitted pollutants in the atmosphere, and (b) existing AQ conditions and what actions should be made in order to avoid hazardous effects to their health condition. The first case is covered in the document and in the hackAIR RS under the concept named "*tips of the day*" and the latter under the "*personalised recommendations*". For both cases, the recommendation text is statically defined in the ontology; no synthesis of recommendation texts will be provided by the hackAIR RS. Nevertheless, the dynamic part of the RS lies in the automated reasoning process, i.e. "*which recommendation(s) should be provided to whom, whenever a new request arrives in the hackAIR system*". Special characteristics and requirements, as well as implementation details of the RS and results are presented in the following sections.

6.1.1 Tips of the day requirements

The first type of recommendations, i.e. *tips of the day*, are useful daily advice on how to improve the ambient AQ and thus to reduce individual's air pollution production. The aim of such messages is to foster environmental-friendly behaviour and to disseminate best practices that could be followed in order to improve air quality or bad practices that should be avoided since they are inherently damaging the atmosphere.

Currently, in the project we have defined a list of 38 tips (see Appendix 8.1) which could potentially be enriched in updated versions of the ontology. The hackAIR tips were created by taking into account that messages should essentially be **informative prompts** which additionally deal with one or more of the following issues: (1) they demonstrate ease of use/acting/execution, (2) they provide alternatives, (3) they offer information of the impacts of past/future habits related to AQ, and (4) they use terms of equivalency for better comprehension. Overall, these tips should not prohibit behaviour, but give alternative ways to trigger behavioural change [hackAIR D2.4, 2017].

Tips are instantiated in the hackAIR ontology under the class `TipsOfTheDay` (sub-class of `Recommendation`). According to schema presented in Section 3.4.1, individuals of this class are connected to the actual text of the tip via the `hasDescription` data property. Tips do not take into consideration any of the personal details stored in user profiles; they are provided randomly to each user upon request. Details behind the rule-based reasoning framework and its implementation for handling and providing tips of the day to the users are given in sub-sections of Section 6.3.2.1.

6.1.2 Personalised recommendations requirements

The second type of recommendations, i.e. *personalised recommendations*, target specific groups of people who are considered by hackAIR experts as the **most vulnerable** ones, either because of their sensitivity to hazardous AQ conditions or because their outdoor activity leads to increased oxygen uptake (e.g. during jogging outdoors), a state which could incrementally be harmful for their health, especially in severe AQ conditions.

Based on the user needs expressed in D2.2 [hackAIR D2.2, 2016] and D2.4 [hackAIR D2.4, 2017] and on results from a performed market analysis, we identified 9 main categories of users who represent the most vulnerable to air pollution individuals and correspond to 9 different user profiles; these are:

- **Infant –** profile with defined age less or equal to 1 year old.
- **Toddler, child or young –** profile with defined age more than 1 and less or equal to 17 years old.
- **Elderly –** profile with defined age more than 60 years old.
- **Pregnant female –** profile with defined value *true* for parameter of pregnancy.
- **Sensitive health –** profile with defined value *true* or specific health disorder (asthma, cardiovascular, etc.) for parameter of health sensitivity.
- **Outdoor job –** profile with defined activity of interest for getting recommendations the *working_activity*.
- **Outdoor sports (general) –** profile with preferred activity any activity that is considered as general in the hackAIR system (biking, jogging, playing tennis, etc.)
- **Outdoor sports walking –** profile with specified preferred activity the *walking_activity*.
- **Outdoor sports picnic –** profile with specified preferred activity the *picnic_activity* (like eating outside).

The above user categories are correlated to relevant user profiles, and thus relevant classes under the class `Person`. Any of the above categories can be assigned either to *direct* or to *indirect users* of the system, except from categories related to persons with age between 0-17 years old, who can potentially be considered only as *indirect users* of the system.

From the semantics behind these classes, it becomes evident that real individual user may belong in more than one of the above categories; hence, he/she belongs to combinations of them, unless these potential combinations make sense. For example, a user cannot belong to both the `Elderly` and `Infant` categories as in real word such a case does not exist. On the other hand, a user may belong to both `PregnantFemale` and `OutdoorSportsWalking` categories or even additionally to `OutdoorSportsPicnic` and `OutdoorJob`.

In order to include such information in the ontology and to represent restriction in combinations, we define a set of disjoint classes per case. Thus, disjoint classes cannot be combined and produce real case user types. The definition of all disjoint classes of each user profile category can be seen in Table 16; disjoint classes are declared with 1 (true) values and non-disjoint classes with 0 (false) values.

Table 16 – A binary matrix representing disjoint classes for each specific user category (sub-classes of Person)

	hackAIR TBox class								
hackAIR TBox class	Infant Person	Mix Child Person[67]	Elderly Person	Pregnant Female Person	Sensitive Health Person	Outdoor Job Person	Sports General Person	Sports Walking Person	Sports Picnic Person
Infant Person	0	1	1	1	0	1	1	1	1
Mix Child Person	1	0	1	1	0	1	0	0	0

[67] `MixChildPerson` class is equivalent to `ToddlerPerson` or `ChildPerson` or `YoungPerson`, which profile categories are considered and treated the same in the hackAIR RS.

Elderly Person	1	1	0	1	0	0	0	0	0
Pregnant Female Person	1	1	1	0	0	0	0	0	0
Sensitive Health Person	0	0	0	0	0	0	0	0	0
Outdoor Job Person	1	1	0	0	0	0	0	0	0
Sports General Person	1	0	0	0	0	0	0	0	0
Sports Walking Person	1	0	0	0	0	0	0	0	0
Sports Picnic Person	1	0	0	0	0	0	0	0	0

The representation of disjoint classes in the ontology give us the basis to define **all possible combinations** of types of user profiles, for which recommendations should be carried out by the hackAIR RS. In Table 17, we present an example of how a single class (here, class `ElderlyPerson`) is combined with all of its non-disjoint classes to result into meaningful complex profiles.

Table 17 – Indicative example of all possible combinations of the class ElderlyPerson with its non-disjoint classes

Combinations per two (2)			
#1	ElderlyPerson	SensitiveHealthPerson	
#2	ElderlyPerson	OutdoorJobPerson	
#3	ElderlyPerson	SportsGeneralPerson	
#4	ElderlyPerson	SportsWalkingPerson	
#5	ElderlyPerson	SportsPicnicPerson	
total	**4 cases**		
Combinations per three (3)			
#6	ElderlyPerson	SensitiveHealthPerson	OutdoorJobPerson
#7	ElderlyPerson	SensitiveHealthPerson	SportsGeneralPerson
#8	ElderlyPerson	SensitiveHealthPerson	SportsWalkingPerson
#9	ElderlyPerson	SensitiveHealthPerson	SportsPicnicPerson

#10	ElderlyPerson	OutdoorJobPerson	SportsGeneralPerson
#11	ElderlyPerson	OutdoorJobPerson	SportsWalkingPerson
#12	ElderlyPerson	OutdoorJobPerson	SportsPicnicPerson
#13	ElderlyPerson	SportsGeneralPerson	SportsWalkingPerson
#14	ElderlyPerson	SportsGeneralPerson	SportsPicnicPerson
#15	ElderlyPerson	SportsWalkingPerson	SportsPicnicPerson
total	10 cases		

Combinations per four (4)				
#16	Elderly Person	SensitiveHealth Person	OutdoorJob Person	SportsGeneral Person
#17	Elderly Person	SensitiveHealth Person	OutdoorJob Person	SportsWalkingPerson
#18	Elderly Person	SensitiveHealth Person	OutdoorJob Person	SportsPicnic Person
#19	Elderly Person	SensitiveHealth Person	SportsGeneral Person	SportsWalkingPerson
#20	Elderly Person	SensitiveHealth Person	SportsGeneral Person	SportsPicnic Person
#21	Elderly Person	SensitiveHealth Person	SportsWalking Person	SportsPicnic Person
#22	Elderly Person	OutdoorJob Person	SportsGeneral Person	SportsWalking Person
#23	Elderly Person	OutdoorJob Person	SportsGeneral Person	SportsPicnic Person
#24	Elderly Person	OutdoorJob Person	SportsWalking Person	SportsPicnic Person
#25	Elderly Person	SportsGeneral Person	SportsWalking Person	SportsPicnic Person

total	**10 cases**					
Combinations per five (5)						
#26	Elderly Person	Sensitive Health Person	OutdoorJob Person	SportsGeneral Person	SportsWalkingPerson	
#27	Elderly Person	Sensitive Health Person	OutdoorJob Person	SportsGeneral Person	SportsPicnic Person	
#28	Elderly Person	Sensitive Health Person	OutdoorJob Person	SportsWalkingPerson	SportsPicnic Person	
#29	Elderly Person	Sensitive Health Person	SportsGeneral Person	SportsWalkingPerson	SportsPicnic Person	
#30	Elderly Person	OutdoorJob Person	SportsGeneral Person	SportsWalkingPerson	SportsPicnic Person	
total	**5 cases**					
Combinations per six (6)						
#31	Elderly Person	Sensitive Health Person	Outdoor Job Person	Sports General Person	Sports Walking Person	Sports Picnic Person
total	**1 case**					

Every row in the table corresponds to **all different** `Person` categories a **single user** may belong to. For example, *combination #27* describes an individual user who has set into his profile the following characteristics: (i) his age that corresponds to the class `ElderlyPerson`, (ii) he belongs to sensitive groups (`SensitiveHealthPerson`), (iii) he usually works outdoors (`OutdoorJobPerson`), (iv) he has some type of preferred activity of general type (`SportsGeneralPerson`) as defined in the hackAIR ontology, and finally, (v) he also prefers going for picnic (`SportsPicnicPerson`). Overall, this user is an *Elderly* & *Sensitive* & *OutdoorJob* & *SportsGeneral* & *SportsPicnic person*, according to the different user classes defined in the hackAIR ontology and supported by the recommendation system.

Similarly to this example, we combined all non-disjoint classes between them (in a set of two or up to six together), and by excluding all disjoint classes from pairing between them, we resulted in a total number of **112 meaningful (unique) combinations**. These combinations define the exact number of different use cases that the RS will support. The number of possible unique combinations, per class and per number of combined classes, is summarised in list below:

- **InfantPerson** – total: 1, Combination:

  per 1
  1

- MixChildPerson – total: 16, Combinations:

per 1	per 2	per 3	per 4	per 5
1	4	6	4	1

- ElderlyPerson – total: 32, Combinations[68]:

per 1	per 2	per 3	per 4	per 5	per 6
1	5	10	10	5	1

- PregnantFemalePerson – total:32, Combinations:

per 1	per 2	per 3	per 4	per 5	per 6
1	5	10	10	5	1

- SensitiveHealthPerson – total: 16, Combinations:

per 1	per 2	per 3	per 4	per 5
1	4	6	4	1

- OutdoorJobPerson – total: 8, Combinations:

per 1	per 2	per 3	per 4
1	3	3	1

- SportsGeneralPerson – total: 4, Combinations:

per 1	per 2	per 3
1	2	1

- SportsWalkingPerson – total: 2, Combinations:

per 1	per 2
1	1

- SportsPicnicPerson – total: 1, Combinations:

per 1
1

From basic to complex profiles

The idea behind user-profile driven decision support is to provide relevant recommendation(s) to the users upon request, with respect to the existing AQ and to the types of user classes where the individuals are classified according to their profile specifications. Since a user may belong to more than one user categories, we result in the aforementioned "112 combination list" (previously presented) that describes all possible combinations for which recommendations should be produced.

In cases where the user belongs to multiple categories, it would be inefficient to produce one single recommendation that integrates all special requirements that the user categories specify. Recalling the previous example, where a user is considered as an *Elderly* & *Sensitive* & *OutdoorJob* & *SportsGeneral* & *SportsPicnic person*, no single recommendation would be sufficient; instead, we split the recommendation into multiple messages, with respect to each different aspect of the user-profile: some aspects give emphasis to the ***health sensitivity*** of a person[69] while other aspects give emphasis to the ***actual activity***[70] that is planned to be performed. Overall, an aspect related to the sensitivity of an individual's health has greater impact in decision support; this aspect will be the basis for all other decisions proposed

[68] Detailed possible combinations for class `ElderlyPerson` are presented in Table 17.

[69] In the hackAIR ontology, user classes named `InfantPerson`, `MixChildPerson`, `ElderlyPerson`, `PregnantFemalePerson`, `SensitiveHealthPerson` and possible combinations of them carry information with respect to the health sensitivity of a person.

[70] In the hackAIR ontology, user classes named `OutdoorJob`, `SportsGeneral`, `SportsWalking` and `SportsPicnic` carry information related to the actual activities of a person for which he/she requests for recommendation.

by the system, whenever a recommendation for an activity should be proposed. Furthermore, aspects related to activities could be separated as being single requests for recommendation.

Keeping the aforementioned definitions and findings in mind, we conclude with a list of **basic-combination recommendations** (i) for all users that belong to only one of the 9 main user profile categories, and (b) for users described from all possible unique combinations per two of non-disjoint classes, where both classes describe sensitivity aspects (giving thus emphasis to the weight of sensitivity) or the one class describes a sensitivity aspect and the other a preferred activity. All other combinations should be considered as complex ones recommendation results should be split into more than one suggestion from the list of basic combination recommendations.

The process of defining *basic* and *complex* combinations of user classes gave the following results: we have 27 basic combinations and 87 complex combinations which are treated as multiple single cases. For the support of the recommendation process we additionally created:

- relevant classes that describe the basic user cases where the individual may belong (see sub-classes of `CombinedCategoriesPerson` presented in Figure 14, and
- relevant recommendation messages were defined and populated in the ontology, with respect to the four different AQ levels (`very good`, `good`, `medium`, `bad`); the list of all recommendation messages defined in the hackAIR ontology as personalised recommendations, is given in Appendix 8.2.
- rules that handle the classification process of the user into relevant basic categories;
- rules that infer relevant recommendation(s) to the user with respect to their user profile and to the existing AQ conditions.

The remaining user profile scenarios (112 minus 27 basic cases = 85 complex ones) will be covered by following the process described next (see Figure 15).

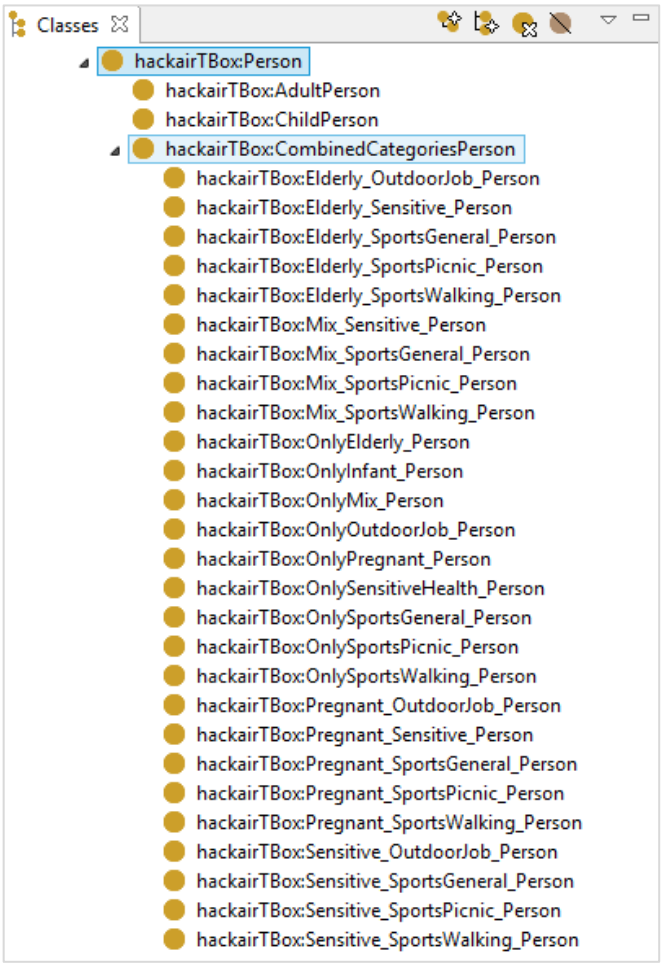


Figure 14 – Sub-classes of CombinedCategoriesPerson for covering the 27 basic user profiles for which single recommendations are provided (screenshot from TopBraid Composer)

Figure 15 – Division of complex profile into basic profiles for assigning relevant recommendations.

In a *complex profile*, where the user belongs to more than one classes of `Person` type and, also, the profile is not represented in the ontology by only one of the 27 basic categories, a splitting should be made properly to fit efficiently with the list of all available personalised recommendations stored in the ontology. The division is made by taking into account the fact that in the complex profile representation, the leftmost class has the highest weight in the recommendation process and thus plays the leading role for recommendation provision and decision support. In Figure 15, the `ElderlyPerson` is the class with the highest weight value for the represented complex profile; while playing the leading role, this will be combined with the remaining classes of the profile in order to match into recommendations of the *basic-combination recommendations* list. As a result, the user with the aforementioned profile will get 4 recommendations, one per basic case into which he/she belongs, i.e. for: (i) Elderly & Sensitive health, (ii) Elderly & OutdoorJob, (iii) Elderly & SportsGeneral, and (iv) Elderly & SportsPicnic person.

To conclude, users of the hackAIR system may be provided with one or more personalised recommendations, depending on their specified profile: if the user belongs to multiple groups for which a single recommendation cannot be assigned, then multiple recommendations will be returned by downgrading the complex profile into multiple base combinations.

We have already populated the hackAIR ontology with all the basic personalised recommendations as instances (individuals) of the class `LimitExposureRecommendation` (sub-class of `Recommendation`). An instance of that type is asserted into the relevant(s) instances of user profile types via the property `isAssignedToPersonCategory`, and also into a specific level of AQ index via the property `isAssignedToAQIndex`. Details about the schema have already been presented in Section 3.4.1 and visualised in Figure 6. The dynamic handling of user profile characteristics and the assignment of the proper recommendation message(s) to the user rely on the hackAIR rule-based reasoning framework, the characteristics and implementation of which is presented in relevant sub-sections of 6.3.

6.2 Reasoning techniques: State of the art

Semantic reasoning is the process of deriving facts and inferring logical consequences from a set of asserted facts or axioms stored in a knowledge base [Berners-Lee, 1998]. The derived facts are called “implicit knowledge” of the ontology. The software tools that are capable of performing semantic reasoning are called *semantic reasoners* or *reasoning engines*.

6.2.1 DL Reasoning

In the case of OWL ontologies, the underlying formalism is based on Description Logics (DL) that are equipped with logically grounded semantics [Baader, 2003], with the main notions being concepts representing sets of objects, roles representing relationships between objects, and individuals representing specific objects. Additionally, DL involves well-defined, powerful reasoning services, for which efficient, sound and complete reasoning algorithms with well understood computational properties are available. The list of most popular, open-source, state-of-the-art implementations includes:

- FaCT++ [Tsarkov and Horrocks, 2006], a tableaux-based reasoner for expressive DL. It employs a wide range of performance enhancing optimisations, including techniques such as absorption, model merging, ordering heuristics and taxonomic classification.
- HermiT [Motik et al., 2009], an OWL reasoner based on a novel hypertableau calculus. This calculus addresses performance problems due to nondeterminism and model size.
- Pellet [Sirin et al., 2007], an open-source, Java-based, OWL-DL reasoner with extensive support for reasoning with individuals (including nominal support and conjunctive query), user-defined datatypes, and debugging support for ontologies.

- Racer [Haarslev et al., 2012] (previously known as RacerPro), a knowledge representation system that implements a highly optimized tableau calculus.

The main reasoning tasks that the above tools can undertake are: (a) *subsumption*: compute all the subclass relationships among the classes (e.g. if concept A subsumes concept B), (b) *consistency*: check if the assertions in a KB have a model (satisfiability), (c) *realization*: compute the instance class memberships (e.g. the set of instances that belong to a certain concept). The aforementioned reasoning tasks are handled within the hackAIR project with the utilisation of Pellet, as being a widely used, lightweight and computational efficient reasoner, compliant to the hackAIR RS requirements.

6.2.2 Rule-based Reasoning

OWL's expressive limitations do not allow for complex reasoning. For instance, OWL is not able to capture relationships between a composite property and another property; a typical example is the composition of the 'parent' and 'brother' properties and the 'uncle' property. This need to model complex relationships beyond the expressiveness of OWL has led to the combination of ontologies with rules, namely, the addition of an extra layer of rule-based reasoning (i.e. extending ontologies with first-order rules) over the above reasoning schemes, capable for more advanced, rule-based inference. Two such paradigms are *SWRL* (Semantic Web Rule Language) [Horrocks et al., 2004] and *SPIN* (SPARQL Inferencing Notation) [Knublauch et al., 2011]. In this deliverable we are working with the latter, due to its higher expressiveness.

SPIN[71] is a well-established standard to represent SPARQL rules and constraints on Semantic Web models. The basic idea of SPIN is to link ontology classes with SPARQL queries that define constraints and rules formalising a specific behaviour of class members. Thus, SPIN allows the implementation of validation layers over ontologies, constraint checking with closed world semantics and automatic raising of inconsistency flags when currently available information does not fit the specified integrity constraints. SPIN takes into account the semantics in the ontology and helps to leverage the fast performance and rich expressivity of SPARQL for various application purposes. SPARQL queries can be stored as RDF triples alongside the RDF domain model, enabling the linkage of RDF resources with the associated SPARQL queries, as well as their consequent sharing and reuse.

6.2.3 Uncertainty Reasoning

Uncertainty unavoidably emerges in practical and real-world reasoning applications. The main sources of uncertainty include: (a) incomplete knowledge, like e.g. missing information and non-exhaustive modelling of domain knowledge; (b) imprecise knowledge, like e.g. the time that an event happened can be known only approximately; (c) unreliable knowledge, like e.g. measurements coming from sensors that can be biased or defective.

There have been various proposals for representing uncertainty in ontologies, mostly revolving around extensions of the underlying languages (RDF, OWL, DLs, rules):

- Probabilistic ontologies refer to extending classical ontologies with probabilistic knowledge [Lukasiewicz, 2008], which involves the representation of terminological probabilistic knowledge about concepts and roles (e.g. "Birds fly with a probability of at least 0.95"), and assertional probabilistic knowledge about instances of concepts and roles (e.g. "Tweety is a bird with a probability of at least 0.9").
- Fuzzy DLs [Straccia, 2001] refer to modelling a domain of interest where these relationships and memberships have a degree of truth in [0, 1], thus, whether an instance belongs to a concept is usually not a matter of "yes/no", but a matter of degree of membership. A major difference between this approach and probabilistic

[71] http://spinrdf.org/

ontologies is the fact that uncertainty in fuzzy concepts usually does not get reduced with the coming of new information.

- Non-monotonic Logics [Horty, 2001], where the set of conclusions may either grow or shrink when new information is obtained. This is contrary to monotonic reasoning systems, where the truthfulness of a conclusion does not change when new information is added to the system – the set of theorems can only monotonically grow when new axioms are added.

6.2.4 Design choices in hackAIR

For the task of designing the architecture and functionality of the hackAIR reasoning framework, we selected to implement SPIN rules on top of the ontology to infer new knowledge. SPIN operates directly on RDF so there is no need to transform ontology data into other formats in order to execute the rules. Its robustness covers the need to dynamically compute implicit values (new triples and, thus, new knowledge) based on what's stated in the model. Moreover, SPIN enables the grouping of rules for giving priority or isolating rules that could be or could be not executed under specific circumstances; it also enables the explicit definition of their execution order, supporting thus incremental reasoning at any level of interest.

Another important decision we made during the design of the hackAIR reasoning framework was to support a crisp approach instead of an uncertainty handling (see previous subsection) implementation, due to the following reason: the required decision support within the hackAIR context can be conceived as a classification task, where different recommendations are given to the users according to a specific set of classes in the ontology they belong; for example, if a user belongs to *Class_A* and *Class_B*, then *recommendation_X* will be provided, or if a user belongs to Class_A, *Class_B* and *Class_C*, then *recommendation_Y* and *recommendation_Z* will be provided, with respect to the existing AQ condition. Thus, the targeted application scenarios and their requirements as defined by the project's environmental experts, do not include any uncertainty factors; they are rather considered as fixed IF-THEN rules and any uncertainty embedded in natural language is eliminated due to semantics represented in the ontology. The representation formalism that we follow, with well-defined boundaries and relations between ontology entities, do not accommodate the representation of vagueness in our recommendation system.

SPIN is perfectly suited for such an approach since it provides the required expressiveness for the knowledge representation and the respective inference mechanisms to efficiently deal with the recommendation of user-profile driven recommendations. It was proved also to be a lightweight approach, complying with the needs for fast computations and response times of the reasoning service, avoiding any redundant complexity of the system. Potential use cases that could deploy uncertainty reasoning in future work are the following:

- The additional inclusion of weather related data in the already defined recommendation process; an example case where the AQ is very good and a user with health sensitivities request for decision support with respect to his/her preferred activity (e.g. likes to go to work by bike), the existing recommendation approach would inform positively the user to proceed with his/her daily routine, but such a recommendation wouldn't be valid in case that it's raining outside. A probabilistic or fuzzy approach would deal with this problem of uncertainty more efficiently and would produce different recommendation in case the weather parameter affects the final decision.
- The inclusion of alternative routes -suggestions which would create additional functionality to the defined recommendation process; the knowledge of spatiotemporal data for a wider area of interest would enable the suggestion of alternative routes giving higher probability to those routes where the user will be underexposed to severe AQ conditions.
- The inclusion of alternative indoor activities –suggestions which could potentially enrich the functionality and use-case cover capability of the defined recommendation process; knowing the preferences/daily routine of

the user with respect to specific indoor activities (going to the gym, being at the office, going to indoor playgrounds, shopping at malls, etc.) would enable the provision of alternative recommendations (preference of indoor instead of outdoor activities), in cases when severe AQ conditions barriers staying/acting outdoors.

6.3 hackAIR rule-based reasoning implementation

In order to make use of the SPIN capabilities, we import the *SPIN vocabulary* into our hackairSPIN ontology; the URI of the SPIN schema is <*http://spinrdf.org/spl*> which in turn imports <*http://spinrdf.org/spin*> and all of its components. In the following list, we summarise the most important SPIN modules [W3C, 2014] that were adopted for the hackAIR rule-based reasoning system:

- SPIN rules - The built-in class named `spin:rule`[72] can be used to specify inference rules using SPARQL CONSTRUCT and DELETE/INSERT, in order to create new triples in the ontology and thus enrich the knowledge stored in the schema. Each implemented SPIN rule define s an inference rule that describes how additional triples can be inferred from what is stated in the WHERE clause of the rule.
- Magic properties - The built-in class named `spin:MagicProperty` can be used to dynamically compute property values even if there are no corresponding triples in the actual model. Magic properties may include input arguments and they also have the ability to return multiple results from the SPARQL query process. Our implemented magic properties are mainly SPARQL SELECT queries for fetching all possible instances that belong to a set of specific classes and at the same time they do not belong to a set of other classes. Multiple results that satisfy their queries can then be passed into other SPARQL modules of the RS process.
- SPIN functions - The built-in class named `spin:Function` can be used to isolate common SPARQL patterns, facilitating this way the decongestion, reuse or extension of SPARQL blocks. For more flexibility, they can be parameterised, by specifying input arguments (parameters/variables). In order to use the SPIN Functions module, we create subclasses of the class `spin:Function`. Our implemented SPIN functions are mainly abstract SPARQL SELECT queries that return results needed for incremental inference in the RS module.

For the hackAIR RS we have implemented a two-level approach; the implemented rules run on top of the hackAIR ontology schema (TBox) and assertions (ABox), where: (a) the *first layer of rules* takes advantage of raw data and produces low level derivations, and (b) the *second layer of rules* takes into account previously asserted derivations so as to combine additional data and produce high level interpretations. Each layer contains multiple individual steps, where the inferences from one step are visible and passed to the next step rules. The step-by-step procedure implemented for the hackAIR RS is described in more detail in the following sub-sections.

6.3.1 First Layer Rules – Low-level derivations

In the first layer of the hackairSPIN ontology we have formed a total number of 145 rules. All of these rules take into account the schema (classes, relations) and the populated instances (individuals) in the ontology and produce low level derivations that are passed into the next layer of rules for more complex interpretations. The rules of the first layer are described in more detail below.

6.3.1.1 Age groups

We have implemented a set of 7 SPIN rules that handle the age of the user in order to categorise him/her into one of the 7 age groups (`InfantPerson`, `ToddlerPerson`, `ChildPerson`, `YoungPerson`, `AdultPerson`, `MiddleAgePerson`, and `ElderlyPerson`) that have been specified in the hackAIR ontology; for the defined age ranges per class see Table 4 in Section 3.4.1. In simple words, these rules do the following example mapping: "*if a user*

[72] spin: is the prefix of URI http://spinrdf.org/spin#

is 32 years old, then he/she is an Adult person". Speaking in terms of the ontology, the corresponding rule for the aforementioned example mapping is given in Figure 16.

```
CONSTRUCT {
   ?person a hackairTBox:AdultPerson .
}
WHERE {
   ?person hackairTBox:hasAge ?age .
   FILTER ((?age > 17) && (?age <= 40)) .
}
```

Figure 16 – SPIN rule for assigning a user with age between 17-40 years old into the class AdultPerson

The age of the user is one of the aspects of the user profile that qualify the sensitivity levels of a person, regardless of potential other health problems. For example, in the hackAIR RS, children and elderly persons are considered more sensitive than adult persons, if no health problems are specified.

6.3.1.2 Observation values

As already defined, the hackAIR KB and RS modules need to handle the quantitative values of air pollutant observations that are fed into the system from the Data Fusion Module (see user-triggered processing pipeline in Figure 1). In the ontology we have defined 4 levels of AQ Index (`perfect`, `good`, `medium` and `bad`) that correspond to the scales presented in Table 18, for different environmental data (AOD in general, PM_{10} and $PM_{2.5}$).

Table 18 – Numerical ranges of AQ observations and their corresponding qualitative values

AQ Index name	AOD scale	PM_{10} scale (μg/m^3)	$PM_{2.5}$ scale (μg/m^3)
Very good	≥0 and ≤0.14	≥0 and ≤20	≥0 and ≤10
Good	>0.14 and ≤0.34	>20 and ≤50	>10 and ≤25
Medium	>0.34 and ≤0.44	>50 and ≤70	>25 and ≤35
Bad	>0.44	>70	>35

The conversion of numerical into nominal values has been implemented with a set of 12 (3x4) SPIN rules in the hackAIR ontology. In simple words, these rules do the following example mapping: "*if a location has a 0.52 AOD value recorded, at the time of request, then this location has a bad AOD Index"*. Speaking in terms of the ontology, the corresponding rule for the aforementioned example mapping is given in Figure 17.

```
CONSTRUCT {
  ?location hackairTBox:hasRelatedIndex hackairTBox:AOD_PM10_bad .
}
WHERE {
  {
    ?location a hackairTBox:LocationCity .
  }
  UNION
  {
    ?location a hackairTBox:LocationCountry .
  } .
  ?location hackairTBox:hasEnvironmentalData ?observation .
  ?observation a hackairTBox:AODEnvironmentalData .
  ?observation hackairTBox:hasEnvironmentalDataType hackairTBox:PM_AOD .
  ?observation hackairTBox:hasNumericalValue ?observation_value .
  ?observation_value a hackairTBox:AODValue .
  ?observation_value hackairTBox:hasValueValue ?tmp_value .
  FILTER (?tmp_value > 0.44) .
}
```

Figure 17 – SPIN rule for assigning a "bad" AQ index to an instance of location that has an AOD value above 0.44

6.3.1.3 Activities

This set of rules is for handling information related to activities. In the hackAIR platform, when creating a profile for recommendation, the user may specify a number of preferred activities from a predefined list of available ones. But, not all activities are treated differently in the RS: activities like *working*, *walking* or *eating outdoors* trigger different personalised recommendations' provision to the user; on the other hand, all other specified outdoor activities that have to do with exertion (i.e. swimming, playing tennis, biking, etc.) are treated the same as they result in similar health impacts in case of bad air quality, and they trigger a recommendation that does not imply any specific activity but it is rather more general (e.g. "*You should reduce your outdoor activities today*"). The SPIN rule that assigns to the user the `sports_general_activity` as preferred activity, whenever he/she specifies in the profile any activity except from the 3 ones, is given in Figure 18.

```
CONSTRUCT {
  ?person hackairTBox:hasPreferredActivity hackairTBox:sports_general_activity .
}
WHERE {
  ?person a hackairTBox:Person .
  ?person hackairTBox:hasPreferredActivity ?activity .
  {
    ?activity a hackairTBox:OutdoorActivity .
  }
  UNION
  {
    ?activity a hackairTBox:IndoorActivity .
  } .
  FILTER (?activity != hackairTBox:walking_activity) .
  FILTER (?activity != hackairTBox:picnic_activity) .
  FILTER (?activity != hackairTBox:working_activity) .
}
```

Figure 18 – SPIN rule that assigns the sports_general_activity as preferred activity of the user, under specific assumptions

6.3.1.4 Other user groups

In the current case, we cover different scenarios of users' categorisation into specific classes of type `Person`; the additional categorisation gives the semantics needed in order for the RS to do the proper assignment of

recommendation(s) to each user profile. In order to achieve this, we combine the use of SPIN rules and SPIN magic properties to get the required results.

Thus, we have implemented 122 SPIN rules, from which:

- 10 rules handle property declarations to instances of users and convert them into relevant user classification, according to the schema of the ontology. For example:
  - *"if two instances of type `Person` are linked to each other via the triple `<person_1 hackairTBox:hasRelatedPerson person_2>`, then `person_1` is the direct user and `person_2` is the indirect user"*, or
  - *"if an instance of type `Person` participates in a triple like `<person_1 hackairTBox:isSensitiveTo health_problem_1>` where `health_problem_1` is an instance of `hackairTBox:HealthProblem` type, or `<person_1 hackairTBox:belongsToSensitiveGroup xsd:true>` then `person_1` is also of type `hackairTBox:SensitiveHealthPerson`."*, or
  - *"if an instance of type `Person` participates in a triple like `<person_1 hackairTBox:hasPreferredActivity activity_1>`, where `activity_1` is an instance of either `WorkingActivity` or `WalkingActivity` or `PicnicActivity` then `person_1` is of `OutdoorJobPerson` or `SportsWalkingPerson` or `SportsPicnicPerson` correspondingly. If `activity_1` is any other outdoor activity declared in the schema, then the user is classified as `SportsGeneralPerson`"*. An example SPIN rule of such type can be seen in Figure 19.
- 112 rules and corresponding number of magic properties that enable the classification of the user into one or more of the 27 unique single/combination profiles; the derived categories act as the basis for the provision of personalised recommendation(s) to the user (see subsection 6.3.2.2). For example, in this category, rules implement a scenario like this: "*if the user is pregnant and walking is her preferred activity then she belongs in both classes `PregnantFemalePerson` and `SportsWalkingPerson`. Such categories are mapped into one single category out of the 27 unique combinations, which is `Pregnant_SportsWalking_Person`*". The corresponding rule & magic property for the aforementioned scenario can be seen in Figure 20.

```
CONSTRUCT {
   ?person a hackairTBox:OutdoorJobPerson .
}
WHERE {
   ?person a hackairTBox:Person .
   ?person hackairTBox:hasPreferredActivity ?activity .
   ?activity a hackairTBox:WorkingActivity .
}
```

```
CONSTRUCT {
   ?person a hackairTBox:OutdoorJobPerson .
}
WHERE {
   ?person a hackairTBox:Person .
   ?person hackairTBox:worksOutdoors true .
}
```

Figure 19 – SPIN rules handling cases for categorising an instance of type Person into an OutdoorJobPerson class

```
SELECT ?person
WHERE {
  ?person a hackairTBox:Person .
  ?person a hackairTBox:PregnantFemalePerson .
  ?person a hackairTBox:SportsWalkingPerson .
  FILTER NOT EXISTS {
    {
      ?person a hackairTBox:OutdoorJobPerson .
    }
    UNION
    {
      ?person a hackairTBox:SensitiveHealthPerson .
    }
    UNION
    {
      ?person a hackairTBox:SportsGeneralPerson .
    }
    UNION
    {
      ?person a hackairTBox:SportsPicnicPerson .
    } .
  } .
}
```

(a)

```
CONSTRUCT {
  ?person a hackairTBox:Pregnant_SportsWalking_Person .
}
WHERE {
  ?args hackairSPIN:Pregnant_SportsWalking_Person ?person .
}
```

(b)

Figure 20 – SPIN magic property (a) and SPIN rule (b) implemented in the hackAIR SPIN ontology, for assigning a relevant user into a specific uniquely combined category

6.3.2 Second Layer Rules – Higher-level interpretations

In the second layer of the hackairSPIN ontology we have formed a total number of 109 rules. All of these rules take into account the schema (classes, relations), the populated instances (individuals) in the ontology and the newly inferred knowledge produced by the first layer of rules in order to produce more high level interpretations of the knowledge represented in the ontology. Results from this layer are the final results of the reasoning process and thus the actual recommendations provided to the users. The set of rules the second layer are described in detail below.

6.3.2.1 Tips of the day

We have implemented a single SPIN rule (see Figure 21a) which assigns a random tip to the user, from a set of available instances of class `TipOfTheDay` that are populated in the hackAIR ontology. For random selection, we have implemented a SPIN function named `hackairSPIN:getRandomTipOfTheDay()` that has no input parameters and returns the text (`?tip_message`) of the randomly selected tip, as seen in Figure 21b.

```
CONSTRUCT {
  ?person hackairTBox:isProvidedWithRecommendation ?tip .
  ?tip a hackairTBox:TipOfTheDay .
  ?tip hackairTBox:hasDescription ?tip_message .
}
WHERE {
  ?request hackairTBox:involvesPerson ?person .
  ?person a hackairTBox:Person .
  FILTER NOT EXISTS {
    ?person a hackairTBox:IndirectUser .
  } .
  BIND (BNODE() AS ?tip) .
  BIND (hackairSPIN:getRandomTipOfTheDay() AS ?tip_message) .
}
```

(a)

```
SELECT ?tip_message
WHERE {
  ?tip a hackairTBox:TipOfTheDay .
  ?tip hackairTBox:hasDescription ?tip_message .
  BIND ((((RAND() * RAND()) * 100) - 10) AS ?rand) .
}
ORDER BY (?rand)
LIMIT 1
```

(b)

Figure 21 – SPIN rule (a) and SPIN function (b) implemented in the hackAIR SPIN ontology, for assigning a tip of the day to a user

According to the requirements of the hackAIR RS, there should be a request that involves a user in order to provide him/her with the tip of the day. Also, such tips are assigned only to direct users of the system; indirect users are excluded from that process. Both requirements are reflected in the implemented SPIN rule.

6.3.2.2 Personalised recommendations

A total number of 108[73] rules have been implemented for covering the reasoning task that is related to personalised recommendations provision. The rule-based reasoning process takes into account the results derived from (i) the user classification that is preceded (see sub-section 6.3.1.4) and (ii) the AQ index categorisation according to the current numerical value of the monitored air pollutant for the area of interest (see sub-section 6.3.1.2). Both results are passed as input in the SPIN function named `hackairSPIN:getRandomGPRecommendation()`, which returns the text of a relevant recommendation from a list of populated recommendations in the ontology.

An indicative SPIN rule of this category is given in Figure 22, which assigns to the user, upon request, the recommendation that corresponds to a user profile of an Elderly & OutdoorJob person (`?person a hackairTBox:Elderly_OutdoorJob_Person`) who lives in an area with bad AQ index (`?location hackairTBox:hasRelatedIndex hackairTBox:AOD_PM10_bad`).

```
CONSTRUCT {
   ?person hackairTBox:isProvidedWithRecommendation ?general_personalized_rec .
   ?general_personalized_rec a hackairTBox:LimitExposureRecommendation .
   ?general_personalized_rec hackairTBox:hasDescription ?rec_description .
}
WHERE {
   ?request hackairTBox:involvesPerson ?person .
   ?person a hackairTBox:Elderly_OutdoorJob_Person .
   ?person a hackairTBox:Person .
   ?request hackairTBox:involvesLocation ?location .
   ?location hackairTBox:hasRelatedIndex hackairTBox:AOD_PM10_bad .
   BIND (BNODE() AS ?general_personalized_rec) .
   BIND (hackairSPIN:getRandomGPRecommendation("bad", hackairTBox:Elderly_OutdoorJob_Person) AS ?rec_description) .
}
```

Figure 22 – SPIN rule implemented in the hackAIR SPIN ontology, for assigning a personalised recommendation to a specific user profile, with respect to existing AQ condition

6.4 A use case scenario for rule-based reasoning

For demonstration purposes of the implemented rule-based reasoning mechanism of the hackAIR platform, we use the *Personas scenario I* described in [hackAIR D2.2, 2016]. In order for the RS to infer relevant recommendations to the users, the following steps should be completed beforehand:

(1) The basic characteristics of each involved profile in the scenario are represented through instantiations of specific hackAIR ontology notions; this step has already been performed in Section 3.5.
(2) The PDL has been applied in order to implement all relevant requests that the RS should serve; this step has already been performed in Section 4.3.

[73] The total number corresponds to the rules created for the 27 basic profile categories for which recommendations are stored in the ontology, and for the 4 different AQ levels defined in the hackAIR system.

Then, every request instantiated in the ontology is handled by the rule-based reasoning mechanism separately. For our presented use case we have already populated in the ontology two instances of type `Request` (see Section 4.3): one for Karl (direct user) and his daughter, Anna (related person, indirect user), and one for Stephan (direct user).

Below, we summarise the existing (*before* running the reasoning mechanism) and inferred (*after* running the reasoning mechanism) triples; ontology data is presented in TURTLE format and inferred triples are marked in the following tables with **`bold font`**. Additionally, in text, we refer the types of rules that are responsible for each inferred value and we explain the meaning behind results.

Before performing the analysis of inferred knowledge with respect to the users, we briefly summarise the set of triples related to the location of interest. Berlin is the city where all users involved in the described scenario live. We assume that the current AQ condition in Berlin is poor, and we give a value to the observed environmental data type (`PM_AOD`) equal to 1.2. In Table 19 an air quality index of bad level (`AOD_PM10_bad`) is asserted to the instance Berlin.

Table 19 – Declared and inferred triples regarding Berlin (involvedLocation) and its current AQ observation (involvedEnvironmentalData)

Instance of type **`Location`** for Berlin (involved location)

`hackairABox:Berlin`
`  rdf:type hackairTBox:LocationCity ;`
`  hackairTBox:hasEnvironmentalData hackairABox:AODEnvData_for_Berlin ;`
`  **hackairTBox:hasRelatedIndex hackairTBox:AOD_PM10_bad ;**`
`.`

Instance of type `EnvironmentalData` for Berlin

`hackairABox:AODEnvData_for_Berlin`
`  rdf:type hackairTBox:AODEnvironmentalData ;`
`  hackairTBox:hasEnvironmentalDataType hackairTBox:PM_AOD ;`
`  hackairTBox:hasNumericalValue hackairABox:AODValue_for_Berlin ;`
`.`

Instance of type `Value` for environmental data for Berlin

`hackairABox:AODValue_for_Berlin`
`  rdf:type hackairTBox:AODValue ;`
`  hackairTBox:hasValueValue "1.2"^^xsd:double ;`
`.`

As mentioned in Table 20, Karl is categorised according to his age as type of `ElderlyPerson`; the rule responsible for such classification is the one declared in first layer (L1) of hackAIR rules, related to the age groups (described in Section 6.3.1.1). Karl is also inferred to be a `DirectUser` since he is the person declared in the request (via the propery `involvesPerson`, which connects an instance of `Request` to an instance of `Person` type); the rule responsible for this classification is included in L1 of hackAIR rules, related to the users and user groups (Section 6.3.1.4). Karl does not specify any preferred activity in his profile, thus no rules related to activities (Section 6.3.1.3) are triggered. For the latter categorisation of the user, rules related to user groups together with relevant SPIN magic properties, infer

that Karl is also of `OnlyElderly_Person`, which is one of the 27 possible basic user categories (details in Section 6.1.2); recommendation will be provided according to this type of `CombinedCategoriesPerson` (sub-class of class `Person`).

Table 20 – Declared and inferred triples regarding user named Karl

Instance of type **`Person`** for Karl (direct user)

```
hackairABox:Karl
  rdf:type hackairTBox:DirectUser ;
  rdf:type hackairTBox:ElderlyPerson ;
  rdf:type hackairTBox:OnlyElderly_Person ;
  rdf:type hackairTBox:Person ;
  hackairTBox:hasAge 63 ;
  hackairTBox:hasGender hackairTBox:male ;
  hackairTBox:hasLocation hackairABox:Berlin ;
  hackairTBox:hasRelatedPerson hackairABox:Anna ;
  hackairTBox:isProvidedWithRecommendation [
      rdf:type hackairTBox:LimitExposureRecommendation ;
      hackairTBox:hasDescription "You should reduce prolonged outdoor activity.
Otherwise take your medication with you (if any).";
    ] ;
  hackairTBox:isProvidedWithRecommendation [
      rdf:type hackairTBox:TipOfTheDay ;
      hackairTBox:hasDescription "Did you know that aggressive driving produces
up to five times more toxic emissions than normal!" ;
    ] ;
.
```

Karl is provided with both types of hackAIR recommendation: (1) a tip of the day informing that "Did you know that *aggressive driving produces up to five times more toxic emissions than normal!*" is assigned randomly to Karl, regardless of his profile or of existing AQ condition in Berlin; rules of the second level (L2) related to tips (Section 6.3.2.1) combined with relevant SPIN function, perform the random selection and assignment of this type of recommendation to Karl, who is the direct user of the hackAIR system, and (2) a personalised recommendation that is specified in the ontology as relevant to Karl's user profile, i.e. ElderlyPerson only; this message prompts to him: "*You should reduce prolonged outdoor activity. Otherwise take your medication with you (if any)*". This message makes clear to the user that the AQ condition in his area is poor, and for this reason Karl, who is an elderly and potentially more vulnerable than a young person, should avoid any unnecessary activities outdoors.

Concerning Anna, as mentioned earlier, she is an indirect user of the system; Karl created an additional profile for her to get recommendations with respect to her characteristics/needs. Since Anna is related to Karl via the property `hasRelatedPerson`, a rule in L1 related to users is responsible for assigning this additional instance of type Person as involved person in the request (see inferred triple in Table 21). In the same layer (L1) and by taking account the same relation populated in the ontology (<`hackairABox:Karl hackairTBox:hasRelatedPerson hackairABox:Anna`>), a rule related to users classifies Anna as an instance of class `IndirectUser`.

Table 21 – Declared and inferred triples regarding indirect user named Anna

Instance of type `Request` for Anna (indirect user)

```
hackairABox:request_from_Karl
  rdf:type hackairTBox:Request ;
  hackairTBox:involvesEnvironmentalData hackairABox:AODEnvData_for_Berlin ;
  hackairTBox:involvesLocation hackairABox:Berlin ;
  hackairTBox:involvesPerson hackairABox:Anna ;
  hackairTBox:involvesPerson hackairABox:Karl ;
.
```

Instance of type `Person` for Anna (indirect user)

```
hackairABox:Anna
  rdf:type hackairTBox:AdultPerson ;
  rdf:type hackairTBox:IndirectUser ;
  rdf:type hackairTBox:Person ;
  rdf:type hackairTBox:PregnantFemalePerson ;
  rdf:type hackairTBox:Pregnant_Sensitive_Person ;
  rdf:type hackairTBox:Pregnant_SportsGeneral_Person ;
  rdf:type hackairTBox:SensitiveHealthPerson ;
  rdf:type hackairTBox:SportsGeneralPerson ;
  hackairTBox:availableMOT hackairTBox:bike ;
  hackairTBox:availableMOT hackairTBox:public_transport ;
  hackairTBox:hasAge 32 ;
  hackairTBox:hasGender hackairTBox:female ;
  hackairTBox:hasLocation hackairABox:Berlin ;
  hackairTBox:hasPreferredActivity hackairTBox:biking_activity ;
  hackairTBox:hasPreferredActivity hackairTBox:sports_general_activity ;
  hackairTBox:isPregnant "true"^^xsd:boolean ;
  hackairTBox:isProvidedWithRecommendation [
      rdf:type hackairTBox:LimitExposureRecommendation ;
      hackairTBox:hasDescription "It's not safe to train outdoors today." ;
    ] ;
  hackairTBox:isProvidedWithRecommendation [
      rdf:type hackairTBox:LimitExposureRecommendation ;
      hackairTBox:hasDescription "Limit your outdoor activities." ;
    ] ;
```

```
  hackairTBox:isSensitiveTo hackairTBox:health_problem_Asthma ;
.
```

Based on the L1 rules relevant for age groups, Anna is classified as `AdultPerson`. Moreover, applying the L1 rules for activities related data (Section 6.3.1.3) and for user groups, a set of inferred triples resulted, describing in a sequence the following facts:

> *Anna prefers to ride a bike → Anna has preferred activity a* `sports_general_activity` *→ Anna belongs to* `SportsGeneralPerson` *class*

Concerning the pregnancy and health sensitivity related data and by following the schema of the hackAIR ontology, the triggering of L1 rules related to users and user groups result the sequence of assertions presented below:

> Anna is pregnant → Anna is categorised as a *`PregnantFemalePerson`*.
>
> Anna has asthma → Anna is categorised as a *`SensitiveHealthPerson`*.
>
> Anna belongs to *`PregnantFemalePerson`* and *`SensitiveHealthPerson`* and *`SportsGeneralPerson`* → Anna is finally categorised as *`Pregnant_Sensitive_Person`* and *`Pregnant_SportsGeneral_Person`*.

The latter categorisation is the one that plays important role in the recommendation process; classes `Pregnant_Sensitive_Person` and `Pregnant_SportsGeneral_Person` are sub-classes of `CombinedCategoriesPerson` and both of them directly define the basic characteristics for which related recommendations will be provided by the system. Thus, for Anna's profile two recommendation messages are asserted: "*Limit your outdoor activities*" and "*It's not safe to train outdoors today*", corresponding to the two aforementioned classes. It should be noted that no tip of the day is asserted to Anna's profile, since she is an indirect user of the system.

To conclude, concerning the request and profile of Stephan, the same rules of L1 categorise him as: AdultPerson and then SportsGeneralPerson (because of his preferred activities – see Table 22) and finally OnlySportsGeneral_Person, which class will be the basis of the proposed recommendation message: "*It's not a good day for outdoor exercise*". When the AQI is bad in a location, the situation may be harmful even for individuals who are not considered that vulnerable to them; for that reason this messages prompts the postponement of any planned outdoor activities as a preventive behaviour against adverse AQ conditions.

Table 22 – Declared and inferred triples regarding user named Stephan

Instance of type `Person` for Stephan (direct user)

```
hackairABox:Stephan
  rdf:type hackairTBox:AdultPerson ;
  rdf:type hackairTBox:OnlySportsGeneral_Person ;
  rdf:type hackairTBox:Person ;
  rdf:type hackairTBox:SportsGeneralPerson ;
  hackairTBox:availableMOT hackairTBox:bike ;
  hackairTBox:hasAge 35 ;
  hackairTBox:hasGender hackairTBox:male ;
  hackairTBox:hasLocation hackairABox:Berlin ;
```

```
  hackairTBox:hasPreferredActivity hackairTBox:biking_activity ;
  hackairTBox:hasPreferredActivity hackairTBox:jogging_activity ;
  hackairTBox:hasPreferredActivity hackairTBox:sports_general_activity ;
  hackairTBox:isProvidedWithRecommendation [
      rdf:type hackairTBox:LimitExposureRecommendation ;
      hackairTBox:hasDescription "It's not a good day for outdoor exercise." ;
    ] ;
  hackairTBox:isProvidedWithRecommendation [
      rdf:type hackairTBox:TipOfTheDay ;
      hackairTBox:hasDescription "Did you know that trees and plants naturally
purify the air? Indoor greenery, a few pots on the balcony or a small garden can
help you breathe deep." ;
    ] ;
.
```

6.5 Technical evaluation of the reasoning framework

In the current task, a technical evaluation of the reasoning framework is presented. We focus on the following aspects to be evaluated:

- the consistency of provided results, by examining if the inferred recommendations comply with those planned to be given to specific types of users through the reasoning process.
- the performance of the framework, in terms of response time to perform the reasoning process, by examining different use cases.

For the first issue, we assigned in two ontology experts (members of the hackAIR consortium) the analysis and debugging of the rule-based reasoning process. Recommendation inferences were examined per each possible use case represented within the context of the hackAIR ontology: simple and complex profiles (belonging in more than one user types – subclasses of `Person` class), direct and indirect users, areas with different levels of AQ, etc. The evaluation showed a deviation from the planned behaviour of the system, in two specific cases (complex profiles) that yielded inconsistent results due to wrong classification results. The error was fixed immediately by correcting the corresponding rules that infer the classification results, and integrated in the final version of the recommendation framework.

For the second issue, we involve different user profiles with different characteristics defined, demonstrating also a scalability in their description, as seen below:

1. a *simple* user profile that contains only information related to the *age* (S-A);
2. a *simple* user profile that contains information related to *age* and *health sensitivity* (S-AHS);
3. a *complex* user profile that contains information related to *age*, *health sensitivity* and one *preferred activity* (C-AHSPA);

4. a *complex* user profile that contains information related to *age*, *health sensitivity* and more than one *preferred activities*[74] (C-AHSPAs);
5. *two involved simple* user profiles of S-A type, the one is a direct and the other is an indirect user (S-A^S-A);
6. *two involved complex* user profiles of C-AHSPAs, the one is a direct and the other is an indirect user (C-AHSPAs^C-AHSPAs);
7. *one simple user* profile of type S-A*, involved with one complex* profile of type C-AHSPAs (S-A^C-AHSPAs);
8. *five simple user* profiles of S-A type (S-A_x5), not related to each other;
9. *ten simple user* profiles of S-A type (S-A_x10), not related to each other;
10. *five complex user* profiles of C-AHSPAs type (C-AHSPAs_x5) not related to each other;
11. *ten complex user* profiles of C-AHSPAs type (C-AHSPAs_x10) not related to each other;

Different user characteristics trigger different rules and generate different number of triples in each relevant step of the reasoning process. Results of the performance evaluation process are summarised in Table 23, where response times and total number of triples generated per case are referred. In our examples, we populate instances of type Person with the aforementioned described details, together with instances of type Request in order to trigger the recommendation process. For the shake of brevity, we consider that users have the same location of interest and the same AQ level at the time of request; this decreases the number of populated triples and of inferred ones. The evaluation process was operated on a computer with the following characteristics: *Intel® Core™ i5-4690K, x64-based processor running at 3.50GHz, with 16GB installed memory (RAM)*.

Table 23 – Triples inferred and response time of reasoning process

#	User profile identifier	# of inferred triples	response time (sec)
1	*S-A*	9	1.35
2	*S-AHS*	10	1.58
3	*C-AHSPA*	16	1.61
4	*C-AHSPAs*	33	1.68
5	*S-A^S-A*	17	1.32
6	*C-AHSPAs^C-AHSPAs*	63	1.88
7	S-A^C-AHSPAs	41	1.47
8	*S-A_x5*	41	1.53
9	*S-A_x10*	81	1.52
10	*C-AHSPAs_x5*	152	1.78

[74] We asserted 4 different activities, 3 of which have targeted recommendations for the activity of interest; these are working, eating outdoors and walking activity. The remaining one is conceived as general activity.

11	*C-AHSPAs_x10*	303	1.89

Generally, the hackAIR RS's performance is impacted by the following parameters: (i) the technical characteristics (processor/memory speed, operating system, etc.) of the system that hosts the RS module, (ii) the capacity of the reasoning engine (here, SPIN Inferencing Engine[75] integrated in TopBraid Composer software), and (iii) the complexity of rules implemented/executed. Since parameters (i) and (ii) remain constant within the experiments, it becomes obvious that any differentiation in times depends on the complexity of rules performed each time.

For all use cases, the response time of the recommendation process ranges from 1.32 to 1.89 seconds, and the number of new inferred triples ranges from a few ten to a few hundred triples, according to the initial statements of each examined scenario. Regarding single user profiles examined that are independent between them (meaning one request for recommendation per profile), the number of inferred triples is a multiplicity of the number of profiles examined; we assume that all simple profiles have the same characteristics. The same findings stand for the complex profiles (i.e. *C-AHSPAs* request corresponds to 33 new triples, and *C-AHSPAs_x10* requests correspond to 303 new triples, a.k.a. almost 10 times the number of triples of one single complex profile). Another fact that is of great interest is that the response time is not drastically affected by the number of examined profiles and of inferred triples. It seems that the proposed recommendation approach is very promising for serving multiple requests at reasonable response times. Nevertheless, this has to be examined in real case scenarios (pilots), when the RS module is planned to be integrated in the hackAIR platform (M20 of the project); at that point, additional parameters will affect the performance of the RS, with most important being the availability and speed of the internet connection/communication between the user and the server.

Finally, a user-centered evaluation of recommendations (both tips and personalised advice messages) is planned to be performed during the pilots (after M20) to measure the satisfiability of the proposed recommendations with respect to users' needs and characteristics.

[75] http://topbraid.org/spin/api/

7 Conclusions and Future work

In this document, we described the ontological framework developed within the hackAIR project, responsible for the representation and integration of heterogeneous data (user profile-, environmental-, recommendation- related information) in order to support personalised recommendation services. The ontology-based implementation concerns two interconnected modules of the hackAIR platform:

- the knowledge base (KB), i.e. the storage module of environmental and user-specific data, according to the concepts and semantic relations described in the ontology schema, and
- the reasoning system (RS), i.e. the rule-based reasoning mechanism that runs on top of the abstract (*schema*) and populated (*individuals*) data of the KB, in order to interpret the existing relations, produce new knowledge and thus infer recommendations to the users, with respect to their profile characteristics and existing air quality conditions.

Within the T4.2, the following issues have been accomplished:

- The creation of the first stable version of the hackAIR ontology, a multi-layered approach that includes the representational schema of the involved information, the base for the assertion of user profile and environmental fused data, and the overall rule-based reasoning mechanism for decision support services.
- The development of a RESTful web service for the orchestration of user profile and environmental fused data, and their dynamic population on the ontology-based KB at run time (i.e. when the user submits a request for recommendation). The problem (*request*) is structured efficiently, via the automated process of transforming data delivered from the hackAIR User profile and Data fusion modules into proper ontological concepts, with the adoption of the presented problem description language (PDL).
- The design and implementation of complex decision rules, with the adoption of well-known, ontology-compliant standard named SPIN. The hackAIR SPIN rules have a predefined sequence of execution in order to achieve incremental reasoning, i.e. low level derivations produce higher level interpretations.
- The specification of the characteristics of user profile categories that are considered as vulnerable in severe air quality (AQ) conditions, and for whom specialised recommendations will be supported. More specifically, we target into elderly persons, people with health sensitivities, pregnant women, children, outdoor sports enthusiasts, people who work outdoors, etc., thus including profiles of people with increased daily exposure or increased potential harm that is especially affected by poor air quality.
- The handling of simple and complex hackAIR profiles: people belonging into more than one category, individuals who claim for recommendation for additional profiles, etc.
- The definition of two types of recommendations: (i) tips of the day, i.e. general suggestions on alternative ways of living and keeping levels of air pollution at a minimum, and (ii) personalised recommendations, i.e. messages for up-to-date environmental updating, with respect to user-profile preferences/characteristics and to existing AQ conditions.

The semantic representation and reasoning framework of the current deliverable will be of significance to the final services of WP5 ("*Development of the hackAIR platform*"). Moreover, the outcome of this task (T4.2) will be updated in T7.3 "*hackAIR support services and methodology update*", based on the feedback from *Pilot operation and evaluation* (WP7). Additional future works include the following contributions:

- Based on the multilingual character of the hackAIR platform, recommendations should be additionally provided in the language of the pilot areas. For that reason, tips of the day and personalised recommendations' messages will be translated into the German and Norwegian languages and also integrated

in the hackAIR ontology. Ontology refinement for the support of the multilingual approach and for the proper inference of recommendations according to the user's language will be accomplished until M20 of the project plan and results will be reported in WP5 "*Development of the hackAIR platform*".

- The creation of the Recommendation and decision support module. This module will realise the reasoning techniques of the current deliverable, in a common framework, in order to provide decision support to the user. Its development will be based on open source reasoning frameworks, such as Apache Jena and will be the interface between the platform (and more specifically, the user profile, data fusion and visualisation modules) and the knowledge base. The API will serve as the mechanism for triggering the rule inference process of the recommendation system. Integration actions with the first version of the hackAIR platform will be settled within WP5 "*Development of the hackAIR platform*", until M20 of the project plan.
- A user-centered evaluation of the reasoning system, and more specifically the assessment of textual recommendations provided to the users, will be conducted by actual users during the pilot operation of the hackAIR platform. Results of this process may lead to refinement of recommendation texts in the ontology, no later than M22 and within the scope of WP7 "*Pilot operation and evaluation*".

References

- [AI^3, 2009] AI3 Adaptive Information, Adaptive Innovation, Adaptive Infrastructure, (2009). *The Fundamental Importance of Keeping an ABox and TBox Split*. [online] Available at: http://www.mkbergman.com/489/?cat=173 [Accessed 24.04.2017].
- [Baader, 2003] Baader, F. (2003). The description logic handbook: Theory, implementation and applications. Cambridge University Press.
- [Bechhofer, 2009] Bechhofer, S. (2009). OWL: Web ontology language. *Encyclopedia of Database Systems*, pp. 2008-2009. Springer US.
- [Bernaras et al., 1996] Bernaras, A., Laresgoiti, I. and Corera, J. (1996). Building and reusing ontologies for electrical network applications. *Proceedings of the European Conf. on Artificial Intelligence (ECAI'96)*, pp. 298-302.
- [Berners-Lee, 1998] Berners-Lee, T. (1998). Semantic web road map.
- [Bontas et al., 2005] Bontas, E. P., Malgorzata, M. and Tolksdorf, R. (2005). Case Studies on Ontology Reuse. *Proceedings of the International Conference on Knowledge Management (IKNOW05)*, Vol. 74.
- [Brank et al., 2005] Brank, J., Grobelnik, M. and Mladenić, D. (2005). A survey of ontology evaluation techniques. *Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005)*, Ljubljana, Slovenia.
- [Brickley and Miller, 2014] Brickley, D. and Miller, L. (2014). FOAF Vocabulary Specification 0.99. *Namespace Document 14 January 2014*, Paddington Edition.
- [Buitelaar and Cimiano, 2008] Buitelaar, P. and Cimiano, P. (2008). Ontology learning and population: bridging the gap between text and knowledge, Vol. 167, Ios Press.
- [Dzbor and Motta, 2006] Dzbor, M. and Motta, E. (2006). Study on integrating semantic applications with magpie. *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, pp. 66-76, Springer Berlin Heidelberg.
- [Falco et al., 2014] Falco, R., Gangemi, A., Peroni, S., Shotton, D. and Vitali, F. (2014, May). Modelling OWL ontologies with Graffoo. *European Semantic Web Conference*, pp. 320-325. Springer International Publishing.
- [Fernandez et al., 1997] Fernandez, M., Gomez-Perez, A. and Juristo, N. (1997). METHONTOLOGY: From Ontological Art Towards Ontological Engineering. *AAAI Technical Report* SS-97-06, pp. 33-40.
- [Fielding and Richard, 2002] Fielding, Roy T. and Richard N. Taylor. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)*, Vol. 2(2), pp. 115-150.
- [Fiorelli et al., 2010] Fiorelli, M., Pazienza, M. T., Petruzza, S., Stellato, A. and Turbati, A. (2010). Computer-aided Ontology Development: an integrated environment. *New Challenges for NLP Frameworks 2010 (held jointly with LREC2010)*.
- [Gangemi et al., 2002] Gangemi, A., Guarino, N., Masolo, C., Oltramari, A. and Schneider, L. (2002). Sweetening ontologies with DOLCE. *International Conference on Knowledge Engineering and Knowledge Management*, pp. 166-181. Springer Berlin Heidelberg.
- [Gangemi et al., 2005] Gangemi, A., Catenacci, C., Ciaramita, M. and Lehmann, J. (2005). A theoretical framework for ontology evaluation and validation. *Proceedings of the Semantic Web Applications and Perspectives (SWAP)*, 2nd Italian Semantic Web Workshop, Trento, Italy.

- [Gangemi and Presutti, 2009] Gangemi, A. and Presutti, V. (2009). Ontology design patterns. *Handbook on ontologies*, pp. 221-243. Springer Berlin Heidelberg.
- [Gruberm, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, Vol. 5 (2), 119-220.
- [Grüninger and Fox, 1994] Grüninger, M. and Fox, M.S. (1994). The role of competency questions in enterprise engineering. *Workshop on Benchmarking – Theory and Practice*, pp. 22-31. Springer US.
- [Haarslev et al., 2012] Haarslev, V., Hidde, K., Möller, R. and Wessel, M. (2012). The RacerPro knowledge representation and reasoning system. *Semantic Web*, Vol. 3(3), pp. 267-277.
- [hackAIR D2.2, 2016] hackAIR Consortium (2016). Deliverable 2.2: User and technical requirement analysis, August 2016.
- [hackAIR D2.4, 2017] hackAIR Consortium (2017). Deliverable 2.4: Report on co-creation of services, June 2017.
- [Han et al., 2008] Han, L., Finin, T., Parr, C., Sachs, J. and Joshi, A. (2008). RDF123: from Spreadsheets to RDF. *Proceedings of the 7th Int. Semantic Web Conference*, pp. 451-466, Springer Berlin Heidelberg.
- [Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member submission*.
- [Horty, 2001] Horty, J. F. (2001). Nonmonotonic Logic. *The Blackwell Guide to Philosophical Logic*. Blackwell
- [Jarrar and Meersman, 2008] Jarrar, M. and Meersman, R. (2008). Ontology Engineering - The DOGMA Approach. *Advances in Web Semantics I*, LNCS Vol. 4891, pp. 7-34.
- [Knublauch et al., 2011] Knublauch, H., Hendler, J. A. and Idehen, K. (2011). SPIN - overview and motivation. *W3C Member Submission*.
- [Lenat and Guha, 1989] Lenat, D.B. and Guha, R.V. (1989). Building large knowledge-based systems; representation and inference in the Cyc project. Addison-Wesley Longman Publishing Co., Inc.
- [Lukasiewicz, 2008] Lukasiewicz, T. (2008). Expressive probabilistic description logics. *Artificial Intelligence*, Vol. *172*(6), pp. 852-883.
- [Maedche and Staab, 2001] Maedche, A. and Staab, S. (2001). Ontology learning from the semantic web. *IEEE Intelligent Systems*, Vol. 16(2), pp. 72-79.
- [Missier et al., 2013] Missier, P., Belhajjame, K. and Cheney, J. (2013). The W3C PROV family of specifications for modelling provenance metadata. *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 773-776. ACM.
- [Mitzias et al., 2016] Mitzias, P., Riga, M., Kontopoulos, E., Stavropoulos, T. G., Andreadis, S., Meditskos, G., and Kompatsiaris, I. (2016). User-Driven Ontology Population from Linked Data Sources. *International Conference on Knowledge Engineering and the Semantic Web*, pp. 31-41, Springer International Publishing.
- [Modica et al., 2001] Modica, G., Gal, A. and Jamil, H.M. (2001). The Use of Machine-Generated Ontologies in Dynamic Information Seeking. *Cooperative Information Systems*, pp. 433-447, Springer Berlin Heidelberg.
- [Motik et al., 2009] Motik, B., Shearer, R. and Horrocks, I. (2009). Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, Vol. 36(1), pp. 165-228.
- [Niepert et al., 2008] Niepert, M., Buckner, C. and Allen, C. (2008). Answer Set Programming on Expert Feedback to Populate and Extend Dynamic Ontologies. *FLAIRS Conference*, pp. 500-505.

- [Noy and McGuinness, 2001] Noy, N.F. and McGuinness, D.L. (2001). Ontology development 101: A guide to creating your first ontology. [online] Available at:

  http://liris.cnrs.fr/alain.mille/enseignements/Ecole_Centrale/What%20is%20an%20ontology%20and%20why%20we%20need%20it.htm

- [Petasis et al., 2011] Petasis, G., Karkaletsis, V., Paliouras, G., Krithara, A., and Zavitsanos, E. (2011). Ontology population and enrichment: State of the art. *Knowledge-driven multimedia information extraction and ontology evolution*, pp. 134-166. Springer-Verlag.

- [Pinto et al., 2004] Pinto, H.S., Staab, S. and Tempich, C. (2004). DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. *Proceedings of the 16th European Conf. on Artificial Intelligence (ECAI)*, pp. 393-397.

- [Poveda-Villalón et al., 2009] Poveda-Villalón, M., Suárez-Figueroa, M.C., García-Delgado, M.A. and Gómez-Pérez, A. (2009). OOPS! (OntOlogy Pitfall Scanner!): supporting ontology evaluation on-line. IOS Press.

- [Raskin and Pan, 2003] Raskin, R. and Pan, M. (2003). Semantic web for earth and environmental terminology (sweet). *Proceedings of the Workshop on Semantic Web Technologies for Searching and Retrieving Scientific Data*, Vol. 25.

- [Rospocher, 2010] Rospocher, M. (2010). An ontology for personalized environmental decision support. *Formal Ontology in Information Systems*, pp. 421-426, IOS Press.

- [Rospocher, 2014] Rospocher, M. (2014). PESCaDO Ontology Documentation. Tech. Rep. Available at: https://dkm-static.fbk.eu/people/rospocher/backup-resources/pescado/PESCaDO_Ontology_Version3.0_Documentation_3.pdf [Accessed 14.06.2017]

- [Straccia, 2001] Straccia, U. (2001). Reasoning within fuzzy description logics. *J. Artif. Intell. Res.(JAIR)*, Vol. 14, pp. 137-166.

- [Sirin et al., 2007] Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A. and Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics: science, services and agents on the World Wide Web*, Vol. 5(2), pp. 51-53.

- [Stadler et al., 2012] Stadler, C., Lehmann, J., Höffner, K., and Auer, S. (2012). LinkedGeoData: A core for a Web of Spatial Open Data. *Semantic Web Journal*, Vol. *3*(4), pp. 333-354.

- [Suárez-Figueroa et al., 2009] Suárez-Figueroa, M., Gómez-Pérez, A. and Villazón-Terrazas, B. (2009). How to write and use the Ontology Requirements Specification Document. *On the move to meaningful internet systems: OTM* 2009. Part of the Lecture Notes in Computer Science book series (LNCS, Vol. 5871), 966-982.

- [Sure et al., 2004] Sure, Y., Staab, S. and Studer, R. (2004). On-To-Knowledge Methodology (OTKM). *Handbook on Ontologies*, pp. 117-132.

- [Swartout et al., 1997] Swartout, B., Ramesh, P., Knight, K. and Russ, T. (1997). Towards distributed use of large-scale ontologies. *Symposium on Ontological Engineering of AAAI*, pp. 138-148.

- [Tartir et al., 2010] Tartir, S., Arpinar, I.B. and Amit P. S. (2010). Ontological evaluation and validation. *Theory and applications of ontology: Computer applications*. Springer Netherlands, pp. 115-130.

- [Tsarkov and Horrocks, 2006] Tsarkov, D. and Horrocks, I. (2006). FaCT++ description logic reasoner: System description. *Automated reasoning*, pp. 292-297. Springer Berlin Heidelberg.

- [Uschold and King, 1995] Uschold, M. and King, M. (1995). Towards methodology for building ontologies. *Workshop on Basic Ontological Issues in Knowledge Sharing, held in Conjunction with IJCAI-95*. Cambridge, UK.

- [Vrandečić, 2009] Vrandečić, D. (2009). Ontology Evaluation, Handbook on Ontologies. *International Handbooks on Information Systems*, pp. 293-313.
- [W3C, 2004] W3C Recommendation (2004). *OWL Web Ontology Language Overview*. [online] Available at: https://www.w3.org/TR/owl-features/ [Accessed 20.04.2017].
- [W3C, 2012a] W3C Recommendation (2012). *OWL 2 Web Ontology Language Document Overview (Second Edition)*. [online] Available at: http://www.w3.org/TR/owl2-overview/ [Accessed 22.05.2017]
- [W3C, 2012b] W3C Recommendation (2012). *OWL 2 Web Ontology Language Primer (Second Edition)*. [online] Available at: http://www.w3.org/TR/owl2-primer/ [Accessed 22.05.2017]
- [W3C, 2014] W3C Recommendation (2014). *SPIN – Modeling Vocabulary*. [online] Available at: http://spinrdf.org/spin.html [Accessed 26.05.2017]

8 Appendix

8.1 hackAIR Tips of the day

Tips of the day are represented in the hackAIR ontology as individuals of `Recommendation` type, and especially of class `TipOfTheDay`. Whenever a new request for recommendation arises, the reasoning system that runs on top of the ontology asserts to the user (`Person` type) a random tip of the day, as a useful daily advice on how to improve the ambient air quality or to reduce individuals' production of air pollution.

When tips are provided to the user through the RS, not any details of the user profile or of request parameters is taken into account. These tips can be conceived as general knowledge that could be useful not only for the direct recipients of the message (hackAIR users) but also for their social environment. This list has been constantly refined by members of the hackAIR consortium and could additionally be enriched with new messages in future versions of the hackAIR ontology.

Table 24 – A list of tips of the day instantiated in the hackAIR ontology

#	Message
1	You could reduce emitted air pollutants if moving by public transport.
2	Want to ride your bicycle? It's not only fun and great exercise; you'll also help to keep the air beautifully clean.
3	Time for a walk? Sweet! Air quality becomes so much better when more people leave their cars at home.
4	Transportation accounts for about 23% of greenhouse gas emissions in Europe. Think about the way you travel in your city!
5	Have you heard of carpool karaoke? What a perfect way to get to work - while keeping the ambient air breathable.
6	Driving slowly on unpaved roads can prevent vehicles from emitting dust.
7	Are you a driver? Did you know that you can reduce your contribution to air pollution by switching off your car motor when idling for more than 20 seconds?
8	To warm up your car, drive slowly the first 5km instead of run. Gentle ride means gentle pollution!
9	Keep your car engine in good condition by performing the regular maintenance.
10	Aggressive driving produces up to five times more toxic emissions than normal!
11	Drive within the speed limits. It works well for both your safety and the environment!
12	Prefer to use energy efficient appliances. This saves you money and it's good for the environment.

13	Keep your water heater at 50°C, and use cold water whenever possible.
14	Switch off the lights when you are not in the room. The room is not afraid of the dark…
15	Unplug electronic devices when not in use. They'll thank you later!
16	Fans are a climate friendly and easy alternative to air conditioners!
17	Did you know that indoor fireplaces are a huge source of indoor air pollution? Limit wood burning.
18	If you have to burn wood in your home, follow useful precautions to reduce pollution.
19	Don't burn wood that is painted. It won't colour the steam, only produce more toxic pollution.
20	Avoid the use of spray products. That smells of... CO2.
21	Manual garden tools are a low-cost alternative to those that run on gasoline. Approved by flowers of all kind!
22	Did you know that a family of four members is responsible for releasing 20 tons of greenhouse gases into the atmosphere each year?
23	Energy consumption in home is the 3rd source of air pollution production. Close heating vents and doors to rooms that you are not using.
24	Hang clothes out in order to dry instead of using a dryer.
25	Whoa, what's that smoke? If you like making a fire in your garden, make sure the wood you use is untreated and dry - your lungs will thank you.
26	Did you know that trees and plants naturally purify the air? Indoor greenery, a few pots on the balcony or a small garden can help you breathe deep.
27	Use natural soy or beeswax candles instead of petroleum/paraffin-based candles.
28	Avoid BBQ when air pollution levels are high. Steak can wait; planet cannot!
29	Recycle as much as you can! Buy products that do not have a lot of packing and that can be recycled.
30	Exercising outdoors early in the morning is a great way to avoid rush-hour air pollution.
31	Use video conferencing for business meetings, when possible, in order to avoid unnecessary travel.
32	Use the hackAIR app to detect and avoid air pollution hotspots!
33	Avoid burning organic waste or garden leftovers. Prefer to compost them!

34	Eat many fresh fruits and vegetables. They help maintain your body's antioxidant reserves which are able to reduce the effects of air pollution.
35	Keep windows closed when outdoor air pollution is high; this way you won't burden the indoor atmosphere.
36	If cooking with gas, use an extractor fan with a filter.
37	Did you know that your exposure to harmful air pollution is greatly increased on main roads with a lot of traffic? If possible, prefer to use secondary roads.
38	If you commute into the city, think about leaving your car at park&ride spots; that way you take away your share from inner city pollution.

8.2 hackAIR Personalised recommendations

Personalised recommendations are represented in the hackAIR ontology as individuals of `Recommendation` type, and especially of class `LimitExposureRecommendation`. In order to provide such types of recommendations to the user, the system takes into account his/her relevant profile, the request made and the existing AQ condition in the area of his/her interest. The defined recommendation messages per possible case are given in the following tables. For the shake of brevity, we present recommendations that evolve only two specific main user categories – Elderly and MixChild (i.e. toddlers, children and young users) – together with all possible combinations of them with their non-disjoint classes of type `Person`; similar recommendation messages are defined and represented in the ontology for all main user categories and their combinations (see Section 6.1.2).

Table 25 – A list of personalised recommendation instantiated in the hackAIR ontology for Elderly class and all possible combinations with the main user categories

Index class	Elderly	Elderly & Sensitive	Elderly & Outdoor Job	Elderly & Outdoor Sports (general)	Elderly & Outdoor Sports (walking)	Elderly & Outdoor Sports (picnic)
Very good	Perfect air for a walk today! The current air quality is ideal for a walk!	*same as Elderly*	Let's work!	Perfect day for training outdoors!	Perfect day for a walk today!	Perfect day for eating outdoors!
Good	Enjoy your usual outdoor activities! Go for it!	*same as Elderly*	You can still work outdoors but be sure to stay alert to our notifications.	You can still exercise outdoors but take your medication with you (if any).	Enjoy your usual outdoor activities!	Let's go on a picnic!

Medium	Consider reducing prolonged outdoor activity.	*same as Elderly*	You should reduce any outdoor work.	You may cancel your outdoor workout for today.	Consider reducing prolonged outdoor activity.	You can still go on a picnic but pay attention for changes in air quality.
Bad	You should reduce prolonged outdoor activity. Otherwise take your medication with you (if any).	*same as Elderly*	It's not safe to work outdoors today.	It's not safe to train outdoors today. In case of breathing problems contact your doctor.	You should reduce prolonged outdoor activity. Otherwise take your medication with you (if any).	You should postpone the picnic for another day.

Table 26 – A list of personalised recommendation instantiated in the hackAIR ontology for MixChild class and all possible combinations with the main user categories

Index class	MixChild	MixChild & Sensitive	MixChild & Outdoor Sports (general)	MixChild & Outdoor Sports (walking)	MixChild & Outdoor Sports (picnic)
Very good	Perfect air for playing outdoors!	*same as MixChild*	Perfect air for training outdoors!	Perfect day for a walk with your child!	Perfect day for eating outdoors!
Good	Let your child enjoy the fresh outdoor air!	*same as MixChild*	Let your child enjoy outdoor sports.	Enjoy a walk with your child!	Let's go on a picnic!
Medium	Consider to reduce prolonged outdoor activity of your child	*same as MixChild*	Your child can still exercise outdoors but pay attention to any signals of breathing problems.	You can still enjoy a walk with your child but be sure to stay alert to our notifications.	You can still go on a picnic with your child but pay attention for changes in air quality.
Bad	Your child should reduce outdoor activities. Otherwise you can visit areas with cleaner air.	*same as MixChild*	You should reduce the outdoor activities of your child.	It's not a good day for a walk.	You may postpone the picnic for another day.

These lists may be refined by members of the hackAIR consortium and additionally be enriched with new messages in future versions of the hackAIR ontology and after the pilot operation and evaluation by the actual users.