Automatic sensor discovery and management to implement effective mechanism for data fusion and data aggregation

Lina Nachabe Ismail

THESE DE DOCTORAT DE

TELECOM SUDPARIS

Laboratoire SAMOVAR

Spécialité : Réseaux et Informatique

Ecole Doctorale : Informatique, Télécommunications et Electronique de Paris

Présentée par

Lina Nachabe Ismail

Pour obtenir le grade de

DOCTEUR de TELECOM SUDPARIS

Découverte et gestion autonomique des capteurs pour une mise en œuvre de mécanismes efficaces de fusion et d'agrégation de données

Soutenue le 6 Octobre 2015

Devant le jury composé de :

BELLATRECHE Ladjel	Professeur des Universités HDR	Rapporteur	National Engineering School for Mechanics and Aerotechnics (ENSMA , Poitiers)
LAFOREST Frédérique	Professeur des Universités HDR	Rapporteur	Telecom Saint Etienne, UJM, Université de Lyon
NGUYEN Thi-Mai-Trang	Maître de Conférences HDR	Examinateur	UPMC - LIP6
FARSEROTU John	Head of the Wireless Program	Examinateur	CSEM - Centre suisse d'électronique et de microtechnique
GIROD-GENET Marc	Maître de Conférences	Encadrant	Télécom SudParis, Institut Mines-Télécom
EL-HASSAN Bachar	Maître de Conférences	Encadrant	Université Libanaise, Faculté du Géni
ZEGHLACHE Djamal	Professeur	Directeur de thèse	Télécom SudParis, Institut Mines-Télécom

2015TELE0021

I dedicate my thesis to my husband and my two precious angels.

ACKNOWLEDGMENTS

Though only my name appears on the cover of this dissertation, many people have contributed to its production. I owe my sincere gratitude to all those who supported me to achieve this work.

First and foremost, I want to thank my advisor Dr. Marc Girod-Genet for his support and guidance all along this thesis. He has been actively interested in my work and always available to advise me. Distances and countries didn't impede the fruitful cooperation between us. I am very grateful for his patience, flexibility, motivation, understanding, encouragement and immense knowledge.

I would also like to give a heartfelt special thanks to my advisor Dr. Bachar El Hassan for giving me the opportunity to do this dissertation and supporting me from the beginning untill the end. You have been a tremendous mentor for me. Thank you for your encouraging, insightful discussions and for all the opportunities that you have offered to me to enhance my knowledge.

I am also grateful to my thesis' director, Prof. Djmal Zeghlashe for his support.

For this dissertation, I would like to thank my review committee members for accepting to read my dissertation. I would also thank my examination committee members. Thank you in advance for your time and constructive comments and suggestions.

I am especially grateful for my students Fadi Aro and Johnny Khawaja for their contribution. Without your efforts and inspirational discussions I could not have been able to validate my work.

A special thanks to my family in Paris who offered me a tender environment during my travel. I will never forget these lovely and enthusiastic moments. For my Aunt thank you for all your support.

As I scramble, I still not find worlds that express my gratitude to my family and my friends. For my family in law, thank you for all the sacrifices that you have made on my behalf. For my Mom and Dad thank you for giving me a solid foundation to conquer life challenges. It is from you and my sister that I have learned hard working, self-confidence, persistence and independency. It is due to these factors I am accomplishing my dissertation now even though all my critical situations.

For my two angels, Lea and Diala, you had been a twinkle in my eyes since you were born. You are the bright light of my life. Your presence motivated and inspired me. I love you so much.

Last but not least, my sincere and deep gratitude goes to my husband Abdulrazzak. You cannot imagine what your presence in my life meant to me. Thank you for supporting me and offering me all the facilities to accomplish this dissertation.

Thanks God for all you have given to me during these three years…

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

RESUME

Actuellement, des descriptions basées sur de simples schémas XML sont utilisées pour décrire un capteur/actuateur et les données qu'il mesure et fournit. Ces schémas sont généralement formalisés en utilisant le langage SensorML (Sensor Model Language), ne permettant qu'une description hiérarchique basique des attributs des objets sans aucune notion de liens sémantiques, de concepts et de relations entre concepts. Nous pensons au contraire que des descriptions sémantiques des capteurs/actuateurs sont nécessaires au design et à la mise en œuvre de mécanismes efficaces d'inférence, de fusion et de composition de données. Cette ontologie sémantique permettra de masquer l'hétérogénéité des données collectées et facilitera leur fusion et leur composition au sein d'un environnement de gestion de capteur similaire à celui d'une architecture ouverte orientée services.

La première partie des travaux de cette thèse porte donc sur la conception et la validation d'une ontologie sémantique légère, extensible et générique de description des données fournies par un capteur/actuateur. Cette description ontologique de données brutes devra être conçue :

- d'une manière extensible et légère afin d'être applicable à des équipements embarqués hétérogènes,
- comme sous élément d'une ontologie de plus haut niveau (upper level ontology) utilisée pour modéliser les capteurs et actuateurs (en tant qu'équipements et non plus de données fournies), ainsi que les informations mesurées (information veut dire ici donnée de plus haut niveau issue du traitement et de la fusion des données brutes).

La seconde partie des travaux de cette thèse portera sur la spécification et la qualification :

- d'une architecture générique orientée service (SOA) permettant la découverte et la gestion d'un capteur/actuateur, et des données qu'il fournit (incluant leurs agrégation et fusion en s'appuyant sur les mécanismes de composition de services de l'architecture SOA), à l'identique d'un service composite de plus haut niveau,
- d'un mécanisme amélioré de collecte de données à grande échelle, au-dessus de cette ontologie descriptive.

L'objectif des travaux de la thèse est de fournir des facilitateurs permettant une mise en œuvre de mécanismes efficaces de collecte, de fusion et d'agrégation de données, et par extension de prise de décisions. L'ontologie de haut niveau proposée sera quant à elle pourvue de tous les attributs permettant une représentation, une gestion et une composition des 'capteurs, actuateurs et objets' basées sur des architectures orientées services (Service Oriented Architecture ou SOA). Cette ontologie devrait aussi permettre la prise en compte de l'information transporter (sémantique) dans les mécanismes de routage (i.e. routage basé information).

Les aspects liés à l'optimisation et à la modélisation constitueront aussi une des composantes fortes de cette thèse. Les problématiques à résoudre pourraient être notamment:

- La proposition du langage de description le mieux adapté (compromis entre richesse, complexité et flexibilité),
- La définition de la structure optimum de l'architecture de découverte et de gestion d'un capteur/actuateur,

- L'identification d'une solution optimum au problème de la collecte à grande échelle des données de capteurs/actuateurs.

ABSTRACT

The constant evolution of wireless technologies and the miniaturization of electrical devices have leads to the massive usage and development of wireless sensor networks (WSNs), which potentially affects all aspects of our lives ranging from military services, passing through e-Health applications, environmental observations and broadcasting, food sustainability, smart home, energy management and smart grid to many other applications. These networks are formed of an increasing number of sensors that measure physical features like temperature, humidity, etc.., actuators that take certain action according to data received from the sensors or through interaction with user like, e.g., regulate the dose of insulin for a patient, and nodes that play the role of coordinator or relay to transmit the data hop by hop to a sink or gateway which gathers the data and send it to the cloud servers for further treatments and alerts generation. These networks are heterogeneous in terms of nodes, energy, computational resources, transferred data and network management, etc. Moreover, these networks are generally implemented as vertical solutions not able to interoperate with each other.

To tackle this heterogeneity and interoperability problems, these nodes, as well as the data sensed and/or transmitted, need to be consistently and formally represented and managed through suitable abstraction techniques and generic information models. Therefore, an explicit semantic to every terminology should be assigned and an open data model dedicated for WSNs should be introduced.

That's why the main aim of this thesis is to specify and formalize an open data model for WSNs in order to mask the aforementioned heterogeneity and interoperability between different systems and applications. This model will also facilitate the data fusion and aggregation through open management architecture like environment as, for example, a service oriented one. The objectives of this thesis can be split into two main objectives:

1) The first objective is to formalize a semantic open data model for generically describing a WSN, sensors/actuators and their corresponding data. This model should be light enough to respect the low power and thus low energy limitation of such network, generic for enabling the description of the wide variety of WSNs, and extensible in a way that it can be modified and adapted based on the application.

2) The second objective is to propose an upper service model and standardized enablers for enhancing sensor/actuator discovery, data fusion, data aggregation and WSN control and management. These service layer enablers will be used for improving the data collection in a large scale network and will facilitate the implementation of more efficient routing protocols as well as decision making mechanisms in WSNs.

Key Words: WSN, Ontologies, WBAN, SOA, Semantic Data Model, Semantic Service Model, Distributed Agents, CoAP.

LIST OF PUBLICATIONS

Refereed Journal Articles & Conference Proceedings

1) L. Nachabe, M. Girod-Genet, and B. El Hassan, "Unified Data Model for Wireless Sensor Network," Sensors Journal, IEEE , vol.15, no.7, pp.3657,3667, July 2015 doi: 10.1109/JSEN.2015.2393951
2) L. Nachabe, M. Girod-Genet, B. El Hassan, and F. Aro, "Applying Ontology to WBAN for mobile application in the context of sport exercises," In Proceedings of the 9th International Conference on Body Area Networks, 29 September 2014
3) M. Hämäläinen, T. Paso, L. Mucchi, J. Farserotu, H. Tanaka, M. Girod-Genet, W. H. Chin, and L. Nachabe, "ETSI TC SmartBAN: Overview of the Wireless Body Area Network Standard" In the Proceeding of the 9th International Symposium on Medical Information and Communication Technology (ISMICT) 2015, 24-26 march 2015, Japan, 2015 (document in press)
4) B. El Hassan, S. Haddad, M. Girod-Genet, L. Nachabe, "Clustering Energy Based Routing Protocol for Wireless Sensor Networks" In the Proceeding of the 2015 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), Costanta, Romania, May 2015.
5) L. Nachabe, M. Girod-Genet, B. El Hassan , J. Jammas, "M-health application for Neonatal Incubator signals monitoring through a CoAP-based multi-agent system", In the Proceeding of Third International Conference on Advances in Biomedical Engineering ICABME'15, 2015, Beirut Lebanon.
6) ETSI TC SmartBAN, "Smart Body Area Networks (SmartBAN); SmartBAN unified data representation formats, semantic open data model and corresponding ontology", TS DTS/SmartBAN-009V1.1.1, April 2015

Contributions

1) Technical Report of Work Item 1. 1 of the ETSI technical comity TC SmartBAN. 2013 till Present.
2) "Service Ontology for Wireless Sensor Network – CoAP," In the 2nd IEEE Lebanon Communications Research Day (IEEE LCRD'14) at NDU Koura, Lebanon, May 3, 2014.
3) "Wireless Body Area Network: Global architecture & Heterogeneity management," In the 7th Lebanese Communication Workshop 2013 (IEEE LCW'13): Communications & Networking for Health Applications at DUT Saida, Lebanon, November 30, 2013
4) "Semantic Ontology for Wireless Sensor Network," In the first IEEE Lebanon Communications Research Day (IEEE LCRD'13) at AUST Achrafieh, Lebanon, June 1, 2013.
5) "Integration of CoAP in BAN", In the E-Health and Tele-medicine Workshop for Elderly Patients: Technologies and Perspectives", IEEE EMBS Lebanon Chapter and IEEE Comsoc Lebanon Chapter, Tripoli Lebanon, 23 May 2015.

ACRONYMS

ANP	Alert Notification Profile
API	Application Programming Interface
ASTM	American Standards for Testing and Materials
BLES	BLE Scanner
CCR	Continuity of Care Record
CEN/TC	Commission for European Normalization/Techical Commity
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
DAML	Darpa Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DIM	Domain Information Model
DL	Description Logic
DoB	Date of Birth
DTLS	Datagram Transport Layer Security
E2E	Continua End-to-End
EDAM	EMBRACE Data and Methods
EDI	Electronic Document Interchanged
EHER	Electronic Health Record
EL	Expression Language
ETSI	European Telecommunications Standards Institute
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GCM	Google Cloud Messaging
GPS	Global Positioning System
HID	Human Interface Device
HL7	Health Industry Level 7
HOGP	HID Over GATT Profile
HR	Heart Rate
HRN-IF	Health Records Network Interface
HTTP	Hyper Text Transfer Protocol
ICT	Information and Communications Technology
IDC	International Data Corporation
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ISO	International Standards Organization
ITS	Intelligent Transportation Systems
JDBC	Java Data Base Connectivity
JSON	Java Script Objects Notation
JSON	JSON Scanner
JW	JSON Writer

LAN	Local Area Network
LAN-IF	Local Area Network Interface
LE	Low Energy
M2M	Machine To Machine
MAC	Media Access Control
MAS	Multi Agent System
MEMS	Micro-Electro-Mechanical Systems
MQTT	Message Queuing Telemetry Transport
MVTU	Ministry of Science, Technology and Innovation
NASA	National Aeronautics and Space Administration
NFC	Near Field Communication
NICTA	National Information & Communication Technology of Australia
NIST	National Institute of Standard and Technology
NW	Node Wrapper
O&M	Observation & Measurements
OBIX	Open Building Information eXchange
OGC	Open Geospatial Consortium
OSI	Open Systems Interconnection
OSWA	Organizational Systems Wireless Auditor
OWL	Ontology Web Language
PAN	Personal Area Network
PAN-IF	Peripheral Area Network Interface
PDA	Personal Data Assistant
PHD	Personal Health Device
PW	Process Wrapper
QL	Query Language
QoI	Quality of Information
QoS	Quality of Service
QUDT	Quantities, Units, Dimensions and Data Types
RAM	Random Access Memory
RDF	Resource Description Framework
RDFS	RDF Schema
REST	Representational State Transfer
RFID	Radio Frequency IDentification
RL	Rule Language
ROM	Read Only Memory
RPM	Remote Patient Monitoring
RSSI	Received Signal Strength Indicator
SAS	Sensor Alert Service
SIG	SIG (Special Interest Group
SMS	Short Message Service
SNOMED	Systematized Nomenclature of Medicine
SOAP	Simple Object Access Protocol

SOS	Sensor Observation Service
SPARQL	Semantic Web Rule Language
SPP	Scan Parameters Profile
SPS	Sensor Planning Service
SSN	Semantic Sensor Network
SUMO	Suggested Upper Merged Ontology
SW	Service Wrapper
SWE	Sensor Web Enablement
SWSN	Semantic Wireless Sensor Network
TAN-IF	Touchable Area Network Interface
TCP	Transmission Control Protocol
TTL	Time to live
UCL	University College London
UDDI	Universal Description, Discovery, and Integration
UDP	User Datagram Protocol
URI	Unique Resource Identifier
URL	Unique Resource Locator
UWB	Ultra Wide Band
W3C	World Wide Web Consortium
WAN-IF	Wide Area Network Interface
WIFI	Wireless Internet for Frequent Interface
WoT	Web of Thing
WSDL	Web Services Description Language
WSN	Wireless Sensor Network
WSSN	Wireless Semantic Sensor Network ontology
WW	WSN Writer
XML	e-Xtended Markup Language

INTRODUCTION

0.1 Thesis Context

Through advances in Micro-Electro-Mechanical Systems (MEMS), wireless technologies, and energy storage techniques, new inexpensive miniaturized sensors/actuators are widely deployed to gather information or control a system. These sensors/actuators are grouped together to form the wireless sensor networks (WSN). These networks potentially affect all aspects of our lives ranging from home automation (e.g. Smart Buildings), passing through electronic Health (e-Health) applications, environmental observations and broadcasting, food sustainability, energy management and Smart Grids, military services to many other applications.

WSNs are formed of an increasing number of sensor/actuator/relay/sink devices capable of communicating through wireless interfaces to deliver the information to the end user. Generally, WSNs' devices (i.e. nodes) are self-organized in clusters and have limited resources (energy, processing and memory) that widely vary from one to another. Furthermore, they are provided by an increasing number of manufacturers which leads to interoperability problems (*e.g., heterogeneous interfaces and/or grounding, heterogeneous descriptions, profiles, models …*). Those WSNs' devices are also sensing different phenomena (temperature, acceleration, light intensity, etc.) and sending various data types (text, Boolean, media, etc.), thus being very heterogeneous (sensed data included). This complicates their unified integration and management within a WSN.

From a network perspective, WSNs are used in various application domains that can differ in their requirement (network lifetime, observation type, topology, communication policies, security, etc.). So, these WSNs are heterogeneous too. In fact, these networks are generally implemented as vertical solutions not able to interoperate with each other.

In parallel to the WSN, the idea of Internet of Things (IoT) has raised. In 2009, K. Ashton (2) introduced the idea of IoT by claiming that if computers can know everything using gathered data, we will reduce cost and time. This means that if computers can take actions without the interaction of people, all decision making will be enhanced. Here comes the idea of machine to machine interaction or M2M.

The benefits of combining the idea of WSN and M2M, as well as integrating WSNs in the IoT, go beyond simply collecting information to offering wide range of services. For example, in a WSN monitoring an elderly in his home, these services can vary from simple monitoring (heart rate, temperature, blood pressure), to reminders (remind the patient to move if he is lying for a long time), alarms and urgent cases handling (send notification if the elderly is falling to the medical staff or his family). However, the main point is how to integrate these heterogeneous

nodes in the IoT. Moreover, how to enable the interoperability between different WSNs to offer shared and combined services.

0.2 Thesis Contribution

Thus, to tackle this heterogeneity and interoperability problems, these WSNs' nodes, as well as the data sensed and/or transmitted, need to be consistently and formally represented and managed through suitable abstraction techniques and generic information models. Therefore, an explicit semantic to every terminology should be assigned, and an open data model dedicated for WSNs should be introduced. SensorML, proposed by Open Geospatial Consortium (OGC) in 2010 (3), has been considered an essential step toward data modelling specification in WSNs. Nevertheless, it is based on XML schema only permitting basic hierarchical description of the data, hence neglecting any semantic representation. Furthermore, most of the researches that have used semantic techniques for developing their data models are only focused on modelling merely sensors and actuators (this is e.g. the case of SSN-XG (4)). Other researches, like e.g. OntoSensor (5), have dealt with data provided by WSNs (i.e. measurements), but without modelling the data type, quality and states.

That is why the main aim of this thesis is to specify and formalize an open data model for WSNs in order to mask the aforementioned heterogeneity and interoperability between different systems and applications. Moreover, an upper service model is conceived to formalize the services offered by these networks. These two models will be combined in a general architecture that will enable the automatic nodes discovery as well as service discovery. This architecture will facilitate the data fusion and aggregation through open management architecture like environment as, for example, a service oriented one. This thesis can therefore be split into two main objectives:

1. To formalize a semantic open data model for generically describing a WSN, sensors/actuators and their corresponding data. This model should be light enough to respect the low power and thus low energy limitation of such network, generic for enabling the description of the wide variety of WSNs, modular and extensible in a way that it can be modified and adapted based on the application.

2. To propose an upper service model and standardized enablers for enhancing sensor/actuator discovery, data fusion, data aggregation and WSN control and management. These service layer enablers will be used for improving the data collection in a large scale network and will facilitate the implementation of more efficient routing protocols, as well as decision making mechanisms in WSNs.

It is worthy to mention that, the thesis is only focusing on proposing solutions for WSNs' interoperability and heterogeneity management, while obviously taking into account low power, low energy and security strong requirements. This is of course, also implying the specification and the design of standardized services and application level enablers, as well as of an open

management architecture like environment, for a generic and secured interaction and access to (as well as control/monitoring of) any sensor/actuator data and entities.

0.3 Thesis Organization

This report is organized as follow.

The first Chapter describes the WSNs, their characteristics, architectures and components. It details the challenges faced in WSNs, especially the aforementioned heterogeneity and interoperability problems. To overcome these problems, semantic annotation techniques are used, in particular semantic ontologies that conceptualize a specific domain (in our case, the WSNs). That is why a detailed description of semantic ontologies, their usage, and validation techniques will also be depicted in Chapter 1. To integrate the WSNs in the IoT, some WSN data communication techniques are presented, as well as the Service Oriented (SOA) and the multi agent architectures.

The second Chapter aims to depict our proposed data model called MyOntoSens ontology. It starts by a general review about the existing WSNs data models. None of the existing ontologies, as well as no combination of the existing ontologies, is able to provide a complete level of reasoning over heterogeneous sensor/actuator data stored in multiple locations. That's why we proposed MyOntoSens Ontology. Being part of the ETSI TCSmartBAN technical comity (within Working Item 1.1), more investigation has been conducted in the E-Health records (EHR) models. To pre-validate our proposed ontology, a mobile application (OntoRun) is developed for monitoring vital signals of a runner during exercises. OntoRun fully relies on our proposed MyOntoSens ontology. It proves the compatibility of MyOntoSens with the Bluetooth LE (Low Energy) profiles, and demonstrates that using ontologies reduces the energy consumption on the mobile sides due to the use of simple rules and light embedded reasoners.

The third Chapter describes our proposed upper service model MyOntoService Ontology. As for the data model, a survey of the existing service model for WSNs and EHR are inspected. Moreover, existing IoT and M2M open models were taken into consideration. Mainly influenced by the ontology of web services OWL-S (6) , we proposed our service ontology. To pre-validate our ontology, a Smart home for elderly people monitoring use case is envisioned, carried out and tested.

The fourth Chapter gives the details of our proposed general architecture. This architecture combines two essential paradigms: SoA and Multi-Agents. This Chapter starts by a review of the existing works about integrating the idea of SoA, as well as distributed agents, within WSN and more specifically in Wireless Body Area Networks (WBAN). Then, a detail explanation about our proposed architecture will be given (the description of agents and how they interconnect). A test-bed of a smart home monitoring elderly in their home has been used to validate and evaluate the overall architecture. This test-bed proves the automatic nodes/services discovery offered by

the architecture. Furthermore, this elderly monitoring at home use case also helps us to determine how and where the agents should be deployed.

Finally, a general summary concluding the work and evaluating the proposed models is depicted. This will also close up with the future works.

CHAPTER 1- BACKGROUND

This Chapter is dedicated to describe WSNs, their domain of applications, their characteristics and constraints. It will, in particular, focus on the heterogeneity and interoperability challenges in WSNs and clarify how to resolve these problems using semantic techniques. Moreover, it details, in section 1.3 some of the newest protocols used in order to integrate WSNs data in the IoT. Finally, it introduces the idea of multi-agent architecture and its usability in WSN.

1.1 Wireless Sensor Networks

The constant evolution of technology in terms of inexpensive and embedded wireless interfaces and powerful chipsets has leads to the massive usage and development of wireless sensor networks (*WSNs*). This potentially affects all aspects of our lives ranging from home automation (*e.g. Smart Buildings*), passing through e-Health applications, environmental observations and broadcasting, food sustainability, energy management and Smart Grids, military services to many other applications. Wireless sensor networks are network of tiny sensing devices, which collaborate with each other via wireless channel in order to gather and process information about some feature of interest. These networks are usually used for monitoring, tracking, or controlling (7)

A deeper investigation about the domain of applications as well as the characteristics, and challenges in WSN are given in the following sections.

1.1.1 Domain of applications

Inanimate objects are interacting with us: we are surrounded by 'things that think' on streets, in homes, in appliances, on our bodies and possibly in our heads. These objects are mainly sensors/actuators deployed in networks to monitor certain phenomena. In fact, the applications of these sensors networks are tremendous.

Starting with environmental observation and forecasting, WSNs can be deployed for volcanic studies and eruption warning system, meteorological observation, fire detection, earthquake studies, water Quality Monitoring, flood, Cyclone and Tsunami Warning System. Moreover, WSNs can be used to prevent from disaster where they are deployed underground mining for to survey deteriorating and/or unstable grounds, and toxic gases. Such systems focus on predicting dangerous environmental phenomena to save our lives (7).

Passing to food sustainability monitoring where the WSN is implemented in the farming area, which is in general unattended area, in particular to prevent plant diseases or manage watering requirement based on soil humidity (i.e. Smart Farming).

With the birth of the Smart City idea, more WSNs' applications have been emerged. Where some WSNs are dedicated to enhance the resources management like Smart Grids or Smart

Cities, others are deployed to monitor the quality of air to enhance the life style of the citizens. Furthermore, vehicle area networks are deployed to enable devices in and around the vehicle to communicate leading to driverless cars. These networks aim to improve driver and vehicle safety by allowing many cars to cooperate with each other and with the sensors deployed in the streets. These applications are gaining a lot of interest (e.g. Intelligent Transportation Systems - ITS). Morgan Stanley estimated that driverless cars will generate $1.3 trillion in annual savings in the United States (8). Another application that is widely adopted is the Smart Home / Smart Building monitoring systems. On one hand, these systems are equipped with sensors to monitor the home's parameters (temperature, humidity, security, etc.) and actuators to automatically control home appliances and equipment's. These Smart Homes are now widely deployed. Acquity Group (Accenture Interactive) estimated that, by 2019, more than 60% of consumers plan to buy connected technology for their homes (8). On the other hand, these Smart Homes encompass a WBAN composed of various medical sensors and actuators located inside or outside the human body, to monitor the vital signals of the resident (e.g. heart rate, body temperature, blood pressure, etc.) or regulate some biological parameters (e.g. the dose of insulin in the blood).

Wireless Body Area Network (WBAN) applications spread to many more usage especially with the explosion of wearable devices. These applications are known as E-health monitoring systems where sensors/wearable devices sense vital signals giving the patient a greater physical mobility and better psychological experience, especially for patients who need assistance (patient under recovery, elderly, handicaps, etc.) and who are no longer compelled to stay in the hospital. With the evolution of smart phones, the wearable devices are capable of communicating with the PDA of the user giving him a huge range of what is called m-health applications. With wearable devices, you can authenticate yourself to open a door or print documents when touching the printer, pay your bills, monitor your health parameters while exercising (speed, cadence, number of calories burned, etc.), play haptic games, etc. Recently, many success stories have been steered. For example, Virgin Atlantic, Japan Airlines and other carriers have tried using smart glasses and watch to improve their check-in service. In addition, Disney invested around $1 billion for MagicBands[1] to get on ride, pay for food and access hotels rooms.

No one can doubt about the importance of the WSNs and how they are changing many industrial sectors like health/medical, environmental, agriculture, industrial, entertainment, safety, and military sectors. Due to the miniaturization of IP-based wireless technology and advancements in communication protocols, the WSN's data are becoming part of the new Internet, the IoT. This new network will encompass sensors, actuators, devices, appliances. IDC (International Data Corporation) forecasts predict that the worldwide market for IoT solutions will become around $7.1 trillion in 2020 (8).

However, the noticeable questions that rise here are:

[1] http://www.economist.com/news/business/21646225-smartwatches-and-other-wearable-devices-become-mainstream-products-will-take-more

- What are the challenges in WSNs?
- How these billions of devices will be integrated in the IoT?
- How this huge amount of data will be handled?

While the first question is answered in section 1.1.2, the two remaining questions are addressed in section 1.3 and section 1.4 that introduce the proposed languages, protocols and architectures to manage the WSN's data and integrate it in the IoT, confronting the challenges of WSN's.

1.1.2 WSN's Architectures

In general, WSNs are ad-hoc networks formed of nodes communicating wirelessly with each other and being able to provide information to a more powerful node which is called sink/data collector/ hub. These nodes are self-organizing static or mobile nodes deployed randomly or in a fixed position depending on WSN's application. WSNs also came in different architectures.

The simplest architecture is the flat architecture where each sensor sends directly the data to a centric node which can belong to the sensor network but considered more powerful, or can be an external entity like PDA or laptop, or can be part of an external WSN. The flat architecture (equivalent to star topology depicted in Figure 1.1) is widely used in WBAN where biomedical sensors collect data and send it to a central device, in general, called Hub (the PDA and in most cases the smart phone of the patient). Although this architecture offers the minimum power consumption, it is limited to short range topologies.

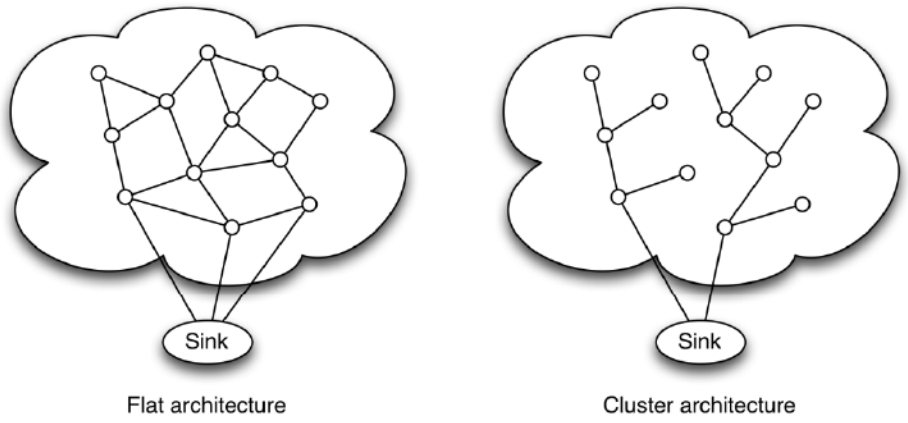

Figure 1.1- Flat architecture (8)

The other types of architectures are the multi-tier architectures which encompass nodes and one or more sink (special nodes that receive data from others). Figure 1.2 depicts an example of a multi-tier architecture where tiers are: the sensors (collect data), the sinks (aggregate data and forward it to a gateway), gateway (decide if the data should be stored or sent to a notification server, or any other server, or displayed on a PDA), PDA (where data can be displayed or heard), and servers (monitoring, controlling, alarms, reminders, etc.).

One of the multi-tier architecture is the data centric architecture used where a wide numbers of sensors are deployed without any global network identifier. Either the sink requests information from a certain node and the information is propagated hop by hop to reach the requested node or a node sends an event query and the sink routes this request. Many routing protocols have been suggested in order to choose the less power consuming route while propagating information. Many times data aggregation techniques are used to decrease the number of routed packets (9).

However, when the number of nodes increases dramatically, the data centric approach will be inefficient due to high latency. In this case, hierarchical architectures have been used. The majority of these architectures use the clustering technique where a number of nodes are grouped in a cluster and one node will collect the transmitted data into the cluster, aggregate it if necessary, then send it to another cluster head or to the WSN's gateway (9). This cluster head is called sink/hub or data collector. ZigBee networks use hierarchical architecture, where a coordinator node will play the role of sink. The coordinator will form a cluster, and collect data. In addition, the ZigBee node can play either the role of router capable of transmitting the data hop by hop after joining the cluster, or the role of end device which can only send/receive data within the cluster (10).

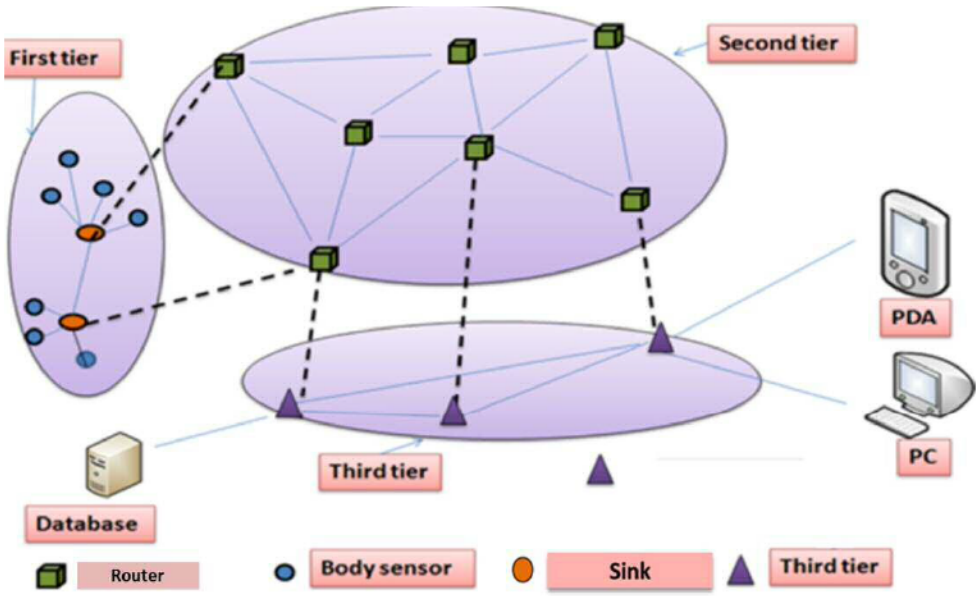

Figure 1.2-Multi-Tier Architecture

More multi-tier approaches have been proposed. For example, the location based architecture where the data is forwarded based on the node's location provided by an internal GPS. Others approaches deal with the problem of mobility in the WSN. Almost all the mobile architectures considered that the sink or some intermediate nodes are mobile (otherwise the problem of mobility will be very complicated). Another routing approach considers the link between two nodes, in general the sink and the node to determine the best path. The link state is defined by the

power cost, latency and delay. For higher reliability, some architecture provides multiple path between two nodes (9).

In summary, wide ranges of WSN's architectures are present and differ from one application to another. Now that we have explored these architectures, a deeper investigation about the characteristics of the nodes' components is given below.

1.1.3 Node's Components & Characteristics

Nodes in WSN are tiny nodes of size and cost. Mainly, these nodes are composed of a transducer unit, processing unit, communication unit, and an energy source unit (11).

The transducer converts the signal from physical phenomena to different physical form. While sensor is an input transducer that responds to a stimulus, actuator is an output transducer that acts in response to a stimulus (11) (12). The physical form is then processed using standard methods and quantity to obtain the measurements. The process used is characterized by its accuracy (how far is the measured value in respect to the real value), precision (variance of a set of measurements), errors (systematic that can be resolved using filters and feedbacks or random caused by noise), drift (low frequency change in a sensor which decreases with electronic aging of the sensor), hysteresis (a linear up and down input to a sensor to produce the result output) and response time (time needed to produce the measurement).

The node's processing unit or CPU is responsible of serving the radio modem, performing computation for application, and executing communication protocol software. In general, the CPU is dedicated for one task. These tiny processors have limited resources of memory (RAM, ROM and flash memory). The most used operating systems are TinyOS and Contiki (13) . For example, TinyOS Mica node has 7.5 Kbits ROM and 600 bits RAM.

In order to transmit the data, the node is equipped with a radio transceiver to send/receive data. Each node can have one or more radio interface belonging to a standard wireless protocol like ZigBee, Bluetooth, NFC, WIFI, etc. These interfaces are characterized by their data rate, frequency range, number of channels, modulation technique, coverage range, and their functioning mode in addition to the power consumed in each operating mode. Typically, there are four modes: receiving, transmitting, idle (ready to receive but not receiving yet), and sleep. The recovery time is the time needed to pass from sleep mode to idle mode. This parameter, if too long, can increase the response time of the overall WSN network and in critical cases, like real time monitoring, can degrade the quality of the network.

Due to their limited size, the energy source of a node should provide as much energy as possible at the smallest cost (large number of nodes are deployed), tiny volume and weight (sometimes Nano-nodes implanted under the skin in some biomedical use cases). In general, these nodes are equipped with non-rechargeable batteries (in general Lithium batteries). Secondary source of energy can be used to recharge the batteries; this is the energy scavenging technique. Other than the usual source of power (wall socket), many approaches try to generate energy form the environment (11). For example, in a WSN where the nodes are monitoring a forest to predict fires, during the presence of the solar energy, the nodes can harvest their energy which will extend their lifetime. Or in WBAN devices can scavenge their power from the human body using body heat and body vibrations.

As we can see, the overall life time of the WSN will be affected by the lifetime of these constrained nodes (constrained in terms of memory, processing and energy's source). Moreover, these nodes differ not only by their resources but also by their functionality. They are playing different roles (nodes/actuators, hub, Sink, relay, etc.) and serving different application adopting various data models. Where some applications require periodic data collection, others are event-driven application where the data is sent once an event is detected (e.g. alarms and reminders) (14). Sometimes, data are sent on request (by external users or sink). Based on WNS's characteristic as well as node's description, we can deduce that although these networks are offering a lot of applications, they have many critical constraints described in section 1.1.4.

1.1.4 WSN's challenges

One of the main challenges in WSN is the topology changes of such networks where hundreds to thousands (or even billions in a context of e.g. Internet of Things and global energy management) of sensors and actuators can be deployed randomly in the sensing fields. These nodes can be moving, which leads to highly dynamic and multiple scales networks being prone to frequent failures. Nodes frequently change their positions and/or locations, and state (active or sleep modes). That is why the WSNs are self-organizing networks. Moreover, additional sensors may be deployed or changed at any time, which poses a need to reorganize the network. Hence, data fusion is essential in these networks where a combination of raw data can lead to improved data. In many cases, the data fusion/aggregation can minimize the power consumed for data transmission (15) . Thus, it is important to provide a solution that enables the automatic sensor discovery and permits advanced techniques for data fusion/aggregation.

In general, large scale WSNs are deployed in unattended areas which are prone to destruction or deterioration. On one hand, nodes may lose battery and die, or may be damaged due to environmental phenomena (fires, storms, etc.). On the other hand, data may be lost due to geographical modification which causes inference or signal attenuation. However, sensed data should be always available regardless the WSN's environment. The installation and maintenance of WSN are considered expensive. Remote management seems to be a good solution for this problem, enabling battery level monitoring and network re-configuration.

At the heart of WSN challenges is the power consumption. These WSNs should last from days to years. The replacement of nodes is so complicated and expensive. Thus, the power consumption should be reduced in order to extend the lifetime of the network. Many researches focused on MAC protocols to resolve the problems of collision that consume inefficiently the power, on less power hungry routing protocol, on data fusion and aggregation to diminish the number of sent data. However, few works focused on how to optimize the power consumption while delivering services offered by these WSNs to external users. Data processing and data transmission from WSNs to external nodes (servers, smartphones, etc.) should be more investigated. Moreover, an essential issue about the repartition of data processing over the nodes should be envisioned.

Another essential challenge in WSN is the heterogeneity in terms of nodes, transferred data and network management. Within the same WSN each node, from different manufacturers, may

require different resources based on its role (node or sink). Where some nodes equipped with large memories are capable of caching and data aggregation, others can only sense data and transmit it to the sink. The wireless interfaces may also differ from one node to another. In general, the sink or gateway are equipped with Bluetooth or ZigBee interface for the inter network communication and a WIFI, 3G/4G or Ethernet interface to be connected to remote monitoring/control/management centre(s). This could be done either through ad-hoc or infrastructure networks (cloud-like included). Furthermore, the data rate, data format and the data type may differ from one application to another and even from one manufacturer to another. The transmitted data can be a text, a value, an image or a video in case of home monitoring for elderly people. Within WSNs, some scenarios need periodic data transmission, others are based on event driven or requests from a gateway. In Body Area Networks (BANs) for example, real time monitoring and alarms, as well as reliability and maximum security level, are also mandatory. Additionally, the lifetime and the Quality of Service (QoS) can vary from one WSN to another. This WSNs' heterogeneity complicates their interworking and thus makes actually impossible the generic interactions with any kind of wireless sensors/actuators (and their sensed data) at the control, monitoring and application levels (e.g. generically reuse and share their data within different applications is still not a reality). In fact, these WSNs are deployed as vertical solutions, not interoperable with each other. Let's differentiate between two types of interoperability:

- Syntactic and structural interoperability that deals with the structure and provenance of information,
- Semantic interoperability: having a common understanding of the information and data sources structure/semantic such that they could be interpreted in a unique way by different WSNs systems.

To really benefit from the potential of WSNs, we should find solutions that mask the problems of heterogeneity and interoperability within the same system, and between different systems, keeping in mind the limited available energy. Furthermore, the evolution of IoT and the birth of WoT (Web of Things) push the WSN's developers to publish sensed data over the Internet. In 2016, 24 million wireless sensors/actuators or sensing points will be used in IoT network (16). It is time to migrate from single and predefined application like WSN to more dynamic, flexible, cooperative WSNs where information can be shared between different systems. Take a look on m-health applications downloaded on your smart phone. Some are used to monitor calories burned during exercises, other are used to assist you for a healthy diet, and other simply monitor your heart rate, etc… Did you ask yourself how many times you entered the same information (e.g. your weight, your height, your gender, etc.)? What if these applications can benefit from one open data model to identify you and retrieve the same data? What if it can revise your clinical visit to a doctor to predict your health problems? Imagine that these diet assistant applications are dynamic enough to propose you meals based on the weather, your mood and ingredients existing in your refrigerator? Can you envision what types of applications can be

emerged if all data sensed by different WSNs cooperate for crucial decision making? This is not an illusion; this can be a reality if we can conceive an open data model enabling data publishing over the internet in a semantic manner. These data can thus be searched and accessed by authorized users like any web page in the actual networks. For illustration, consider that a doctor wants to monitor the heart rate of his patients anytime and anywhere. He just sends his request to a cloud monitoring server, and he totally ignores how this data is technically sensed, processed, transmitted and received on his smart phone. In order to do so, open, flexible, pervasive, dynamic semantic service oriented architecture should be conceived, in a light manner, to offer the users the service they need. Section 1.2 depicts the technologies used to tackle the heterogeneity and interoperability management. Section 1.3 presents the newest techniques for data publishing. Moreover, SoA and multi-agents techniques are introduced in sections 1.4 and 1.5.

1.2 Semantic Ontologies

As already mentioned, one of the major problems facing WSNs is the data/measurements heterogeneity and the lack of interoperability among various devices and systems. Actually, data generated by WSNs are stored in files, databases (relational or non-relational databases) or XML format. Let's consider a patient equipped with a heart rate sensor that sends his/her data to his smartphone where an application is dedicated for heart rate monitoring and calories burned during the day. In addition, let us also consider that he installed an application that tracks his daily activity (speed, how many hours he runs, quality of his sleep, etc.). This patient already has some cardio problems. He/She travelled abroad, he/she attempted a heart attack and arriving to the hospital the doctor should diagnose him. Imagine if this doctor can access the full medical history of this patient, can retrieve the previous measurements taken by the heart sensor and can monitor the previous activities done by the patient. Let's go further, imagine if this doctor can check the medical history of the patient's relative. Evidently, the diagnostic will be much easier and effective. Unfortunately, current mobile applications are not designed to share their data and many few of these applications provide full history of the recorded data. In addition, medical data are not portable, unless the patient has saved it on the cloud for example, and if he is not conscious he could not access it. Thus, the need to find a way to share all these data among these incompatible and independently designed systems is clearly pointed out and quite becoming mandatory. One solution is the use of XML files. The different systems should conceive a common XML schema where the data is structured. But, if a new system intends to share data with these previous systems, a restructuring of the XML schema is required to add new parameters. Moreover, if two solutions are using different vocabularies for the same object, XML techniques cannot capture this redundancy. Or, going back to example, the patient can be named as patient, person, user, etc. Different terminologies are used to describe the same properties, even within the same domain. For that reason, new techniques that deal with the data meaning, not only with the data structure, are needed. These techniques are named Semantic techniques where an entity presents an aspect of the real domain described by metadata (data

about data). In 2004, W3C recommended RDF (Resource Description Framework) language to describe semantic data. RDF is written in XML format and defines a clear set of rules for providing simple descriptive information. It is a method of conceptual data modelling based on triples (subject, predicate, and object). RDFS (RDF Schema) can be used to relate different triples into a common vocabulary (17). Everything defined by RDF is called resource. RDFS defines:

- Classes: can be identified by Internationalized Resource Identifiers (IRI). A class can be sub-class of other class (e.g. relatives a sub class of Users)
- Instance: members of a class (e.g. "Lina" a member of Users Class)
- Literal & Data Type: e.g. string, Boolean, xml, token, etc.
- Properties: that relates the subject resource (Domain) to the object resource (Range). It can be a Data Property, where the domain is a class and the range is a data type, or an object property that links two classes.

RDFS enables the use of sub-classes and sub-properties description. However more semantic description should be added to enhance the understanding of the semantic Model. In that context comes the role of ontologies which can be stored RDFS models with enhanced semantic descriptions.

1.2.1 Ontologies descriptions

Using the formal definition (17), ontologies are "tools for specifying the semantics of terminology system in a well-defined and unambiguous manner". In 2004, W3C recommended the use of OWL language to describe ontologies. The use of OWL paves the way to the integration of semantic interoperable data into the new Web, the WoT. Built on the top of RDFS, OWL adds more description like equality between classes or properties, data types, and properties description (inverse, transitive, etc.). Moreover, OWL permits the usage of predefined ontologies due to the use of imports in its header. OWL came in three flavours: OWL Lite, OWL DL and OWL Full. OWL Lite mainly includes the following descriptions:

- *SameClassAs*, *samePropertyAs*, *sameIndividualAs* (can be used for patient identification where a patient can be described differently into systems but should be considered as same individual), and *differentIndividualFrom*.
- *InverseOf* (if a property P1 is *inverseOf* property P2, than the range of P1 should be the domain of P2 and vice versa), *TransitiveProperty*, *SymmetricProperty*, and *FunctionalProperty* (if P1 is a functional property of a class C1, thus, any individual of C1 may have only one value for P1), *InverseFunctionalProperty*
- *MinCardinality*, *maxCardinality*, Cardinality (The cardinality can be 0 or 1).

In OWL DL a new type enumerated (oneOf) has been added. Moreover, *unionOf*, *complementOf*, and *intersectionOf* properties can be set between classes and individuals. OWL Full permits cardinalities different than 0 or 1, and the statement that classes are disjoint.

Due to the use of these semantic descriptions in OWL, new data can be deduced. For example, if Lina *hasChild* Lea, and *hasChild* is defined as inverse property of *hasParent*, the information "Lea *hasParent* Lina" can be deduced. This is inference. In fact, the most powerful point behind using OWL is the capability of reasoning in order to infer more significant data. Inference can improve the quality of data management in WSNs by discovering new relationships, by automatically analysing the content of the data, or by managing knowledge on WSNs in general. Inference based techniques are also important in discovering possible inconsistencies in the integrated data. The latest version of OWL is OWL 2 released in 2009. Merely, having an ontology language without reasoning facilities to discover non-intended consequences and inconsistency is pointless. We have thus retained OWL 2 DL (Description Logic) to formalize our open data model (MyOntoSens Ontology, refer to Chapter 2) and service model (MyOntoService Ontology, refer to Chapter 3). OWL 2 DL offers decidability by allowing consistency checking and automatic reasoning on Knowledge Base using realistic computing resources in a reasonable time. It can also be extended to support Semantic Web Rule language (SWRL). Let's see now the available reasoners, as well the software and libraries that can be used to create and handle OWL2 DL models.

1.2.2 Reasoners, Inference and Queries

A reasoner engine is responsible for inferring logical consequences from asserted facts or axioms. It is also responsible for classifying, debugging and querying the ontology. Three different reasoning profiles exist: OWL 2 RL (Rule Language) reasoning, OWL 2 QL (Query Language) and OWL 2 EL profile (this profile is based on the EL family of description logics) (18). Many reasoning engines have been proposed in the state of art. Dentler *et al.* (18) presented in their paper a deep comparison between different reasoners. In fact, while choosing a reasoner, what is mandatory is to find a compromise between expressivity and computational complexity. What really enhances the expressiveness of a model is the use of additional Semantic Web Rule Language (SWRL). Now, only Hermit, Pellet and RacerPro (part of SWRL reasoning) reasoners support rule based inference engine. Thus, while conceiving our open data/service models we restricted our choice between Hermit and Pellet reasoner. Thinking about the lifecycle of our proposed ontologies, where individuals will be added from different components and the ontologies will be updated any time, pushes us to choose the reasoner which supports incremental reasoning. For that reason, Pellet incremental reasoner was chosen. The comparison done by Dentler *et al.* shows that Pellet reasoner was able to classify the benchmark ontologies in a reasonable time consuming acceptable memory. Regarding the computational complexity, we believe that a modular ontology can help in reducing the time as well as the memory allocation needed for reasoning. This idea will be proved later on while evaluating our proposed ontologies. The main features of Pellet (19) are :

- Ontology analysis and repair.
- Datatype reasoning.
- Conjunctive ABox query: a query about individuals and their relationship to data and other individuals through datatype or object properties.
- Consistency checker that ensures that our ontology doesn't contain any conflicting facts.
- Satisfiability that verifies for each class if it is possible to have instances.
- Classification of the ontology and creation of the complete hierarchy.
- Realization by finding the most specific classes those individuals belong to.

Pellet supports inverse and transitive properties, cardinality restrictions and data-type reasoning.

After inferring the data, it is time to query it. OWL default query language is Semantic Protocol and RDF Query Language, SPARQL (20). A Query Language allows verification of facts on a Knowledge Base, retrieval of matching statements, and building of new statements. Four types of queries exist (see Figure 1.3):

- SELECT: returns all, or a subset of, the variables bound in a query pattern match
- CONSTRUCT: returns an RDF graph constructed by substituting variables in a set of triple templates
- ASK: returns a Boolean indicating whether a query pattern matches or not
- DESCRIBE: returns an RDF graph that describes the resource found.

SPARQL engine supports (21):

- Optional data matching (although there is no data, the query is still valid).
- RDF data sets combination between different models using URIs that identify each resource.
- The ability to handle incomplete or contradictory information.
- Extraction of XML and RDF graphs.
- Retrieval of inferred data

In summary, the combination of OWL DL with the powerful Pellet reasoner, and the capability of extracting information from different models as well as inferred information, drives any model to an intelligent, flexible, adaptable model that takes into consideration the pervasive environment and process additional data when needed. This is exactly what is required to tackle the problem of heterogeneity and interoperability management in WSNs. Hence, our proposed data and service models are built using OWL 2 DL, reasoned using Pellet reasoner, and queried

using SPARQL engine. We have now to investigate the software used to create and handle OWL ontologies.

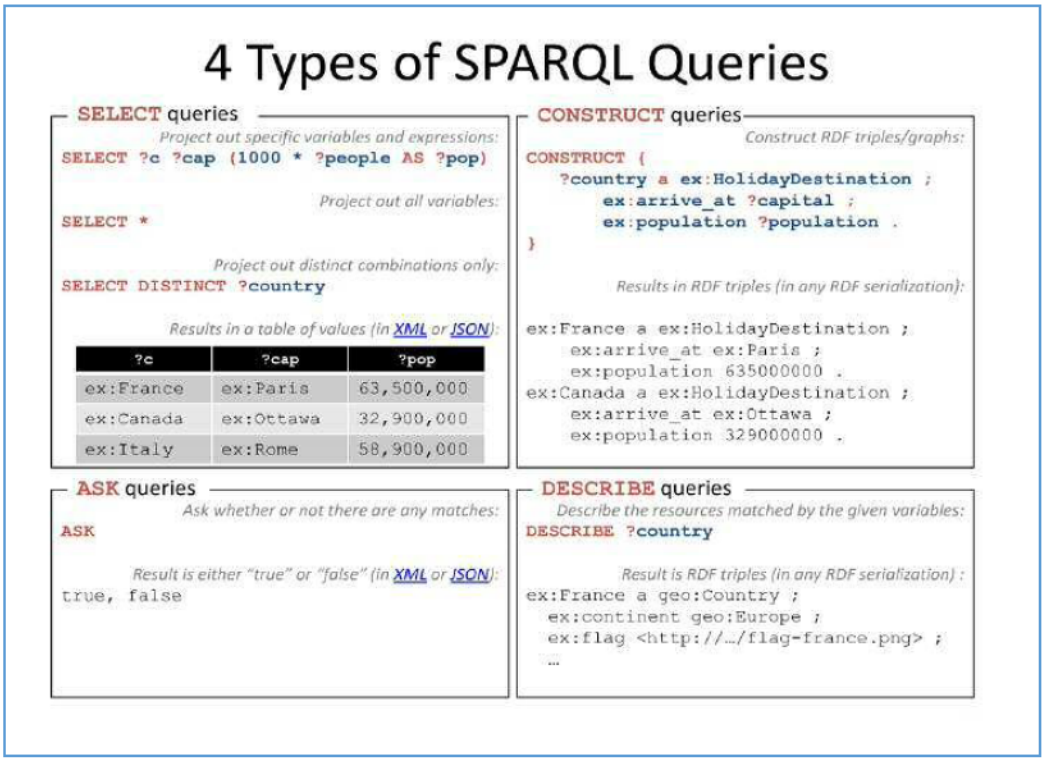

Figure 1.3-SPARQL Types[2]

1.2.3 Ontologies Software

Many programs and environments can be used to design and edit ontologies like Protégé, OilEd, OntoEdit and Jena. To create our ontologies we have used the latest version of Protégé (4.3) for the following reasons (22):

- Is mostly used,
- Can import existing ontologies,
- Supports RDF/OWL structure,
- Supports wide range of reasoners, especially Pellet reasoner,
- Supports SWRL language,
- Supports SPARQL queries,

[2] Adopted from http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet

- Provides clear visualization of classes and properties (OntoGraph plugin),
- Provides clear justification of inferences,
- Has a user friendly interface.

When implementing our use cases, we have chosen Jena API framework written in java. Jena was developed by Brian McBride (23). It provides an interface and classes for RDF manipulation, as well as for ontology management. It also has in memory and persistent storage, which is essential for semantic servers that retain all the created individuals. Jena is designed so that inference engines can be 'plugged' in Models. It supports a rule-based inference engine for reasoning with RDF and OWL data sources. A query engine, compliant with the latest SPARQL specification and capable of retrieving inferred information, is also available within Jena (contrary to Protégé SPARQL engine that only retrieves asserted data). Figure 1.4 depicts the overall structure of the Jena API inference engine. The Model Factory represents the ontology model used to create individuals.

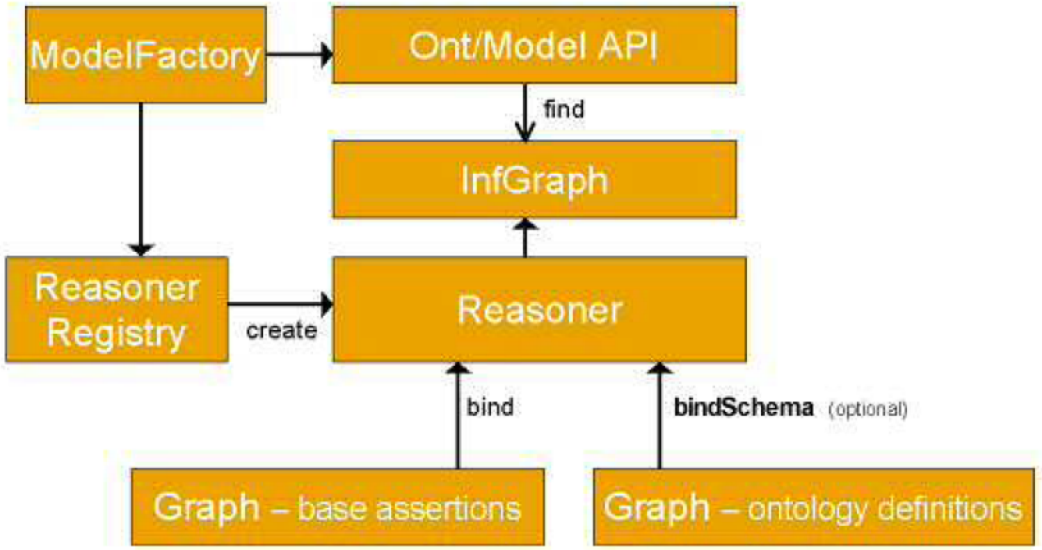

Figure 1.4-Overall Architecture of the Jena API Inference Engine[3]

The Model Interface depicted in Figure 1.4 is used for creating resources, properties literals, for adding or removing statements, for querying a model and for many operations related to combination of models.

To start making a Model, we must use the read method. This method allows the reading of a model from: an "*InputStream*" containing the text model, or the reader that also read the text model, or a URL link where the base model is hosted. Various types of model can be created:

[3] Adopted from https://jena.apache.org/documentation/inference/

RDFS, OWL Lite, OWL DL, and OWL Full. For each OWL Model, the developer can precise the inference model required (no inference, external inference, inference including rules, or transitive inference). When implementing OWL model and inference engine in the smart phone, a lighter version dedicated for Android devices was used: the AndroJena framework. It can be used on Android 4.x smartphones. SPARQL engine is still under development for embedded devices.

In summary, the ontologies are becoming an essential part of the web. Today, search engines are deeply used in biomedical sciences due to the ability to reason and infers over huge datasets, especially in genecology. They are also used in the current intelligent search applications like Google search and Apple Siri. Being one of the most complete OWL editors, we have chosen Protégé 4.3 that enables users to build and edit classes and properties, to import ontologies, to visualize ontologies in different manners, to access reasoning engines, to edit and execute queries and rules, to compare ontologies. We have also retained the Jena API framework for implementing our use cases because it handles RDF processing, it can be used on an Android Smartphone and it provides: persistent in memory storage, rule based inference engine and SPARQL compliant query engine. After developing ontology, it should be validated and evaluated. The following section gives a summary about the existing ontology validation methods.

1.2.4 Ontology validation methods

An overview of existing ontology validation methods is presented. Those models will be investigated here with the objective of selecting the most appropriate one for qualifying our proposed data/service models and associated ontologies.

For qualifying and validating ontology, three approaches have already been defined (24) :

- Evolution-based validation,
- Symbolic-based validation,
- Attribute-based validation.

The first aforementioned validation method consists in observing the evolution of the ontology usage over the time. The original ontology schema is a posteriori compared to all the instances of that ontology that was introduced and used during a given period of time. The retained evaluation criteria are, for example:

- Ontology domain changes. It is introduced for tracking any new knowledge that could have been added to the domain formalized by the ontology,
- Changes in the ontology usage perspectives in a given domain. It is introduced for tracking any changes in ontology usage impacting the ontology conceptualization,

- Ontology specification changes. It is introduced for tracking ontology stability by detection of the number of new objects or attributes that have been introduced in the original ontology graph.

In the context of WNSs, it is clear that this validation method can't be used alone since it is mandatory to validate the specified ontology before its usage in WSNs' applications. This validation method has therefore to be combined with ether a symbolic based and/or an attribute based approach.

The second validation approach is the symbolic one. It is a rule-based evaluation where two rule sets are defined: one containing rules directly coming from the ontology itself, and the other containing rules coming from the user domain. The first set encompasses rules like dissimilarities, matching, inclusion or inheritance between objects and/or attributes. The second set incorporates rules coming from the user domain for identifying conflicting properties in the ontology specifications. This approach seems to be very useful in the context of our thesis where our proposed ontologies should be first validated using reasoners that support both rules coming from the ontology itself (OWL 2DL axioms) and additional rules (defined in SWRL).

Finally, the third approach consists in scanning the ontology graph for measuring the knowledge level through the use of quantitative metrics like, e.g., location and distribution of the ontology objects in the graph.

1.2.4.1 Common used symbolic-based method

While ontologies are used to infer implicit knowledge, the evaluation approach based on rules is essential. Basically, this approach checks for consistency which means that the ontology does not contain any contradictory facts. From literature (25) we can distinguish three consistency's types:

- Structural consistency: to check that the ontology respects the ontology language underlying rules. In OWL language, this means checking for TBox rules. This also means to verify the classes' structures like subclasses, disjoint classes and equivalent classes. For example, a subclass of a class should be a class,
- Logical consistency: to validate that the ontology conforms the logical theory of the ontology. These rules are describes in Abox rules. They allow the verification of the data properties and object properties constraints. The domain and ranges are checked in addition to the values, the equality between properties and the inequality between properties. Moreover, the logical consistency ensures that a functional property should be unique in the ontology model. The inverse properties, reflexive, irreflexive, symmetric and asymmetric properties are tested in this level,
- User-defined: These rules are added by the users. Some inferences can be gathered from the ontology model itself, but others may not be expressible in the ontology language

(usually OWL) and require a more functional representation. Here comes the role of the SWRL,

- As part of the owl rules, the user can determine the maximum cardinality and minimum cardinality for the object properties.

After consistency checking, the statisfiablity (i.e. if it is possible for a class to have instances) is tested. Then, the classification process is executed in order to create the complete class hierarchy. Finally, the realization is executed where most specific classes that individuals belong to are shown. This is the main evaluation approach that we have adopted for our proposed data/service ontologies. It has been carried out using the Pellet reasoner.

1.2.4.2 Common used attribute-based methods

A wide variety of metrics have been proposed in order to evaluate the ontologies in terms of complexity and completeness. Other metrics were also proposed in order to compare between different ontologies and rank them. OntoClean (26) is a framework that helps developers to evaluate their ontologies in order to check for inconsistency and conflicts. The developer should add meta-property (Essence, Rigidity, Identity and Unit) for the predefined properties. A property is tagged as "Essense" if it is essential to the ontology model. A special form of essentiality is rigidity. In this case, the property is essential for all entities. Thus, a property can vary from being rigid, semi-rigid to anti-rigid (i.e. which means not essential to all properties). The other type of meta-property is used to determine the equality between the entities. This is done by indicating the identity of the property. In addition, the developer can designate the unity of a property (if it is a unit property or if it depends of other properties). The unity will help in checking the consistency of the ontology. OntoClean can be used with Protégé. However, OntoClean is considered complex for a developer, especially if he/she is dealing with large ontologies where he/she has to add meta-property for many properties.

The OntoQA tool developed by LSDIS Lab at the University of Georgia (24) is a simpler analysis method that provides a number of metrics that includes individuals, thus it allows for the automatic measurement of ontologies. In OntoQA, metrics (features) are divided into two groups: schema metrics that address the design of the ontology schema, and instance metrics that address the way instances are organized within the ontology. Although we cannot definitely know if the designed ontology correctly models the domain knowledge, metrics in this category indicate the richness, width, depth, and inheritance of an ontology schema design. This is done by calculating:

- The relationship richness. It measures the number of the properties in the selected class that are actually being used in the instance ontology relative to the number of relationships defined for the selected class in the ontology definition,
- Inheritance richness. It shows the number of inheritance relationship from the total number of relations,

- And attribute richness. It indicates the distribution of instance over classes.

The way data are placed within ontology is also a very important measure of ontology quality because it can indicate the effectiveness of the ontology design and the amount of real-world knowledge represented by the ontology. Instance metrics include metrics that describe the KB (Knowledge Base) as a whole, and metrics that describe the way each schema class is being utilized in the KB. These metrics are the class richness, class connectivity, which indicates the number of instances of other classes connected by relationships to instances of the selected class, class importance and the cohesion.

Contrary to the previous methods that are used only by developer, OntoCAT ontology consumer analysis (27) was proposed for being used by ontology developer and user for evaluation purposes. It is composed of two metrics categories: intentional metrics and extensional metrics. Intentional metrics are calculated based on the ontology definition itself, i.e. its classes and subclasses and their properties. Extensional metrics deal with the instances and how effectively the ontology is including the domain knowledge. In other terms, OntoQA schema metrics match OntoCAT intentional metrics, and OntoQA instance metrics match OntoCAT extensional metrics calculated on the ontology occurrences. OntoCAT defines the following groups of metrics (27):

- Intentional size metrics: show the numbers of classes, relationships and properties in a tree or sub-tree,
- Extensional size metrics: show the number of occurrences of classes, relationships and properties,
- Intentional structural metrics: show structural metrics that are similar to size metrics, since they are over the entire intentional ontology, i.e. over all the root trees defined in the ontology (if no concept or class is specified),
- Extensional structural metrics: same as the previous metrics, but on occurrences,
- Summarization metrics: select the object occurrences (for extensional) and classes (for intentional) having the maximum number of in and out links. For intentional, the count of links is the number of subclasses and super classes defined. For extensional, the links are based on the relationships specified for creating its structure.

OntoCAT can be used during the life cycle of the ontology due to the use of extensional metrics that shows the evaluation of the ontology after individuals' creation.

OntoMetric (28) is designed to help the users in choosing the appropriate ontology. It checks the content, the language, the developed methodology, the building tools and the usage cost to decide which ontology is suitable for a certain domain of application. The user should choose reference ontology and, based on the analytic hierarchy process; it decides whether or not to reuse the ontology.

In conclusion, where some evaluation methods were designed to help the users in selecting the adequate ontology, others focus on helping the developers to identify the complexity, the core and the consistency of their ontologies. In fact, while the users look for accuracy, adaptability, extensibility, clarity, and organizational fitness, in certain domain, the developers should provide conciseness (no irrelevant elements or redundant representation of semantics), lightness (minimum allocation of resources in terms of memory and processing), consistency (no contradictions) and flexibility. Thus, we evaluate our ontology using Pellet reasoner, and then estimate its complexity by monitoring the time needed to classify the elements, its memory size, and its processing exigency. More evaluation should be done during the ontology life cycle, e.g. by users during its usage. The ontology should also be ranked in relation to other ontologies covering the same domain. We are still working on implementing our models on large scale use cases and test beds like smart homes dedicated for elderly monitoring.

After exploring the semantic techniques let us now see how WSN's data can be part of the WoT and shared between different devices.

1.3 WSN's data communication protocols

This section gives an overview about data communication protocols that are used for WSNs and that could be retained for our proposed general architecture. Raw data gathered form sensors should be send to a semantic node/server. Moreover, when an external user requests a data, this data should be previously published over the Internet. Many protocols already exist and we are only focusing on commonly used or emerging ones that could be adopted for our proposed architecture detailed in Chapter 4.

Furthermore, WSNs data should be shared by different devices for further treatment. Referring to section 1.1.2, a data collector or hub should collect the data and send it, in case of multi-tier architecture to a local server. Moreover, these data are forwarded, if needed, to multiple cloud servers for diagnostic and advance decision making. Thus, a data communication language should be used to transfer data between different nodes. This language should be interpreted by different devices regardless the software and hardware specifications.

1.3.1 JSON and JSON LD

For a while, XML was the only interoperable data exchange. But, with the need to interchange data in constraint devices, a format was needed to be processed faster. JSON (Java Script Objects Notation) emerged to exchange JavaScript objects. JSON is a simple, completely independent, text format built on two structures: a collection of name/value pairs and an ordered list of values. In most languages, this is realized as an array, vector, list, or sequence. Unconsciously, JSON is lighter than XML because less data are used for describing and sending the same information. However, unlike XML, JSON only supports basic formats (text and numbers), cannot be queried, and does not allow the creation of new data types. Therefore, we should just use JSON for transferring simple data, especially between the constraint/embedded devices (e.g. Smart phone

of the user) and the local server, or between different servers. For example, if the WSN's user is equipped with his/her smartphone and should send his/her name, email, and date of birth, to the server, these data are exchanged using JSON messages as shown below:

```
{"user":[
    {"firstName":"John", "lastName":"Doe",  "email":"  j.h@hotmail.com",  "dob":"22-08-1999"
},
   {"firstName":"Anna", "lastName":"Smith",  "email": a.s@hotmail.com" ,  "dob":"12-03-1987"
};
]}
```

With the linked data, JSON introduced a new format, called JSON-LD (29) , where the value can be an embedded link to another object hosted on another web hosting domain. It is very easy to use it in an environment where objects are identified by their URI like in ontologies environment (Refer to section 1.2.1). Besides data exchanging, integrating WSN data in the WoT also demands the use of data publishing techniques. These techniques are summarized in the following section.

1.3.2 CoAP for data publishing

After collecting the data from sensors, data should be published over the Internet. Many techniques already exist for data publishing, either via the use of database servers where the client can retrieve the data using queries, or through web service techniques. While our main aims are:

- To provide a flexible and extendable solution for any monitoring system,
- To integrate collected data in the IoT in order to be shared by other applications and monitor/control servers (e.g. within a cloud) for data aggregation and management,
- And to mask the heterogeneity between different development platform and software, regardless the programming language and the operation system used.

We choose to publish our data using web service techniques. In fact, web services can be carried out using two types of environments/protocols: SOAP and REST. SOAP defines a set of rules for data communication encapsulated in XML messages. The SOAP server provides certain functions that can be invoked by a client (30).When real-time monitoring is needed, REST environment/architecture is preferable because it is a stateless protocol, with cashing capability, and therefore faster than SOAP.

The use of IP in embedded systems pushes the researchers to view the embedded device as an internet resource that becomes controllable by servers. Each resource is thus identified by a URI (Unique Resource Identifier). The existing technologies for resource controlling over Internet are built on the top of REST (Representational State Transfer) architecture which uses TCP (Transmission Control Protocol) as transport protocol. The CoRE working group proposed

Constrained Application Protocol "CoAP" (31) that uses UDP as transport Layer instead of TCP to decrease the overhead of the TCP protocol (CoAP is more appropriate for wireless network environments).

CoAP includes the following main features:

- Constrained web protocol fulfilling M2M requirements.
- UDP [RFC0768] binding with optional reliability supporting unicast and multicast requests,
- Asynchronous message exchanges,
- Low header overhead and parsing complexity,
- URI and Content-type support,
- Simple proxy and caching capabilities,
- Stateless HTTP mapping, allowing proxies to provide access to CoAP resources via HTTP in a uniform way or for HTTP simple interfaces to be realized alternatively over CoAP,
- Security binding to Datagram Transport Layer Security (DTLS).

CoAP messages are encoded in a simple binary format starting with a fixed 4-bytes header. Figure 1.5 depicts the CoAP message format. Each message is identified by a message ID that must not be re-used within the exchange lifetime (using a configurable parameter EXCHANGE_LIFETIME). The Type field indicates the type of the message which can be one of the following:

- CON: Confirmable message that needs acknowledgement. In this case CoAP implements the reliability feature of TCP, but at the application layer.
- NON: non confirmable message,
- ACK: Acknowledgement message sent as a response to a confirmable message,
- RST: A Reset message indicating that a specific message (Confirmable or Non-confirmable) was received. However, some context is still missing to properly process it.

The code field is composed of 3 bits class (Success(0); response (2); client error response (4), server error response (5)) and of 5 bits providing details. The Token is used to correlate between a request and a response. Options field can be used to indicate if the sent message is Elective, Critical, Unsafe or Safe to forward. The payload, if present, should be prefixed by a fixed one-byte Payload Marker (0xFF) which indicates the end of options and the start of the payload. The payload will hold the data to be transferred.

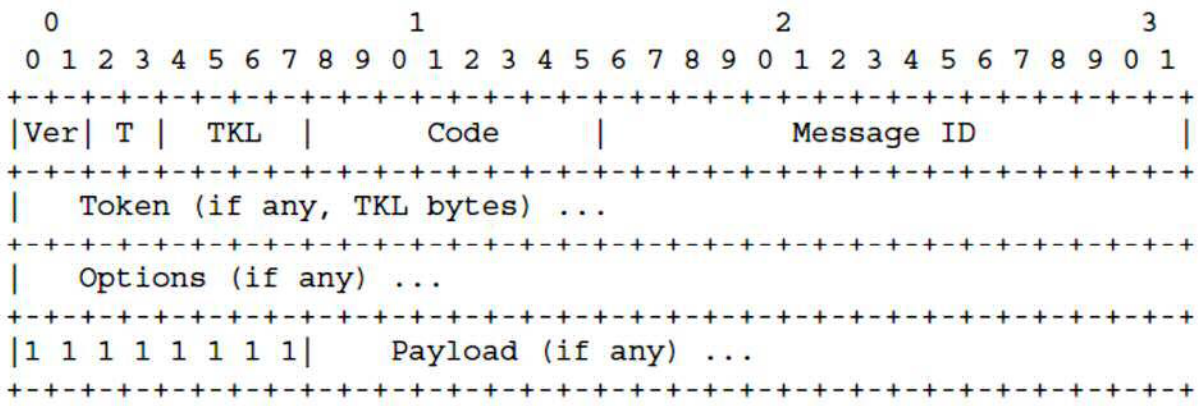

Figure 1.5-CoAP Message (31)

CoAP can be used for client/server applications where a CoAP endpoint plays the role of a "client“ who send CoAP requests to a "server“ which services the requests by sending CoAP responses. A request is carried in CON or NON message. Moreover, Piggy-Backing can be used for acknowledgement. A CoAP request consists of the method to be applied to the resource (GET, POST, PUT, or DELETE), the identifier of the resource (URI), a payload and Internet media, and optional meta-data about the request. The CoAP URI is given as follow:

coap-URI = "coap:" "//" host [":" port] path-abempty ["?" query]

The default port is 5683 is assumed. An argument is often in the form of a "key=value" pair.

CoAP includes a simple caching model determined by response code. The MAX-AGE option indicates the caching lifetime of a resource. In addition, the ETag option can be used in the GET request to give the origin server an opportunity to both select a stored response to be used and update its freshness.

Unlike HTTP, the cache-ability of CoAP responses does not depend on the request method, but on the Response Code. Proxies can be explicitly selected by clients (forward-proxy). In addition, proxies can be inserted:

- To stand in for origin servers (reverse-proxy),
- Or to map from a CoAP request to a CoAP request (CoAP-to-CoAP proxy),
- Or to translate from/to a different protocol (cross-proxy).

Figure 1.6 shows an example where a CoAP proxy is used to convert from HTTP to CoAP request/response.

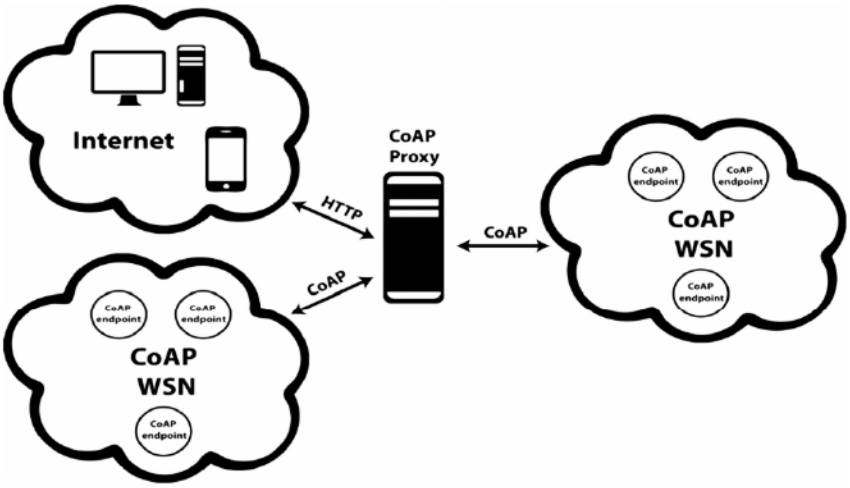

Figure 1.6-Implementation of CoAP proxy (32)

CoAP implements 4 security types:

1- NoSec: There is no protocol level security.

2- PreSharedKey: DTLS (Datagram Transport layer Security) is enabled. A list of pre-shared keys is defined and each key includes a list of which nodes can be used to communicate with.

3- RawPublicKey: DTLS is enabled and the device has an asymmetric key pair without a certificate (a raw public key) that is validated using an out-of-band mechanism. The device also has an identity calculated from the public key, and a list of identities of the nodes it can communicate with.

4- Certificate: DTLS is enabled and the device has an asymmetric key pair with an X.509 certificate that binds it to its Authority Name and that is signed by some common trust root.

In summary, due to the use of JSON techniques, simple data are exchanged between different parties for further treatment and processing (e.g. reasoning). These processed data can be published over the Internet using a multicast and light application protocol like CoAP. WSN data are now part of the WoT empowering the opportunities for advances application where many WSNs can share these data, analyse it and enhance the decision making. WSN data can be seen as services provided by its WSN network (like fall detection server), where the external user requests a precise service independently of the underneath technology used in the network. How to do so? SOA (Service Oriented Architecture) enables to access WSNs as services. Moreover, keeping in mind the dynamicity and the power restriction in WSNs, multi agent architecture seems to be very suitable in order to distribute the processing on nodes, and response to the dynamicity of such networks.

1.4 SOA and Multi-agents Architecture

Service Oriented Architecture (SOA) is an architectural paradigm and discipline to conceive infrastructures permitting the consumers and providers to cooperate via services across different domains of technology (32). The architecture is mainly composed of services that can be deployed in different languages and platform, enabling the interoperability between different systems. In SOA, service providers publish their service in order to be discovered and consumed by service consumer as depicted in Figure 1.7.

Auspiciously, SOA provides agility, which means that it is a loosely coupled solution that supports run-time selection of the requested service in a dynamic and collaborative environment (32). This is exactly what is needed in WSN. Sensors/nodes will play the role of service providers (monitoring service) and users, cloud servers will play the role of service requestors. WSNs development requires changing from monolithically thinking to dynamic, distributed, shared solutions and tightly coupled architectures. WSN nodes publish their data, and any external node, from the web or an external WSN, can request this service. Of course, service brokers are needed to match the requested criteria to the available published services. Monitoring alerts, reminders, controlling are examples of services that can be offered by WSN. Distributed computing is inevitable in deploying WSN solution since these services could not necessarily be computed by the same node. Moreover, the service capability and characteristic will change over the WSN lifecycle. For example, if a GPS is used for a tracking service, the resolution of this service will change based on the location of the user (Indoor or outdoor). Sometimes, the battery level, if drained, affects the accuracy of the measurement. The ideal way is to integrate the SOA in multi agent architecture.

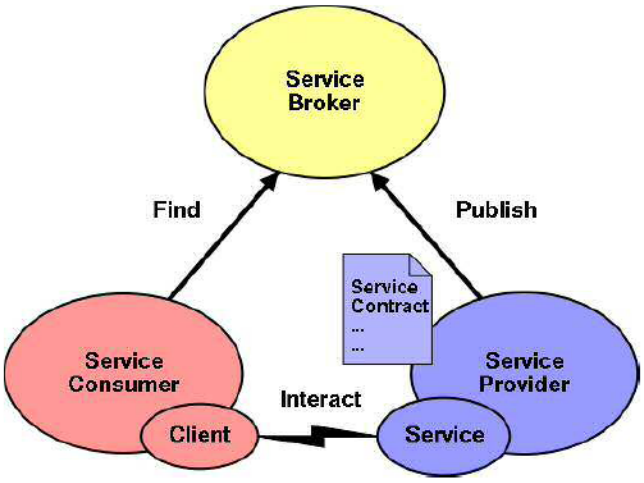

Figure 1.7-Service Oriented Architecture[4]

[4] Adopted from https://www.shawsystems.com/products/loan-lease-servicing/retail/retail-service-oriented-architecture-soa/

One of the major requirements in building modular architecture for WSN is the autonomy. The architecture should be divided in standalone modules offering certain services and capable of interacting with other modules. This is exactly what software agents can do. An agent is a piece of software that can run autonomy to perform certain behaviour. The composition of many agents is called Multi Agent Systems (MAS) (33). The agents interact by passing predefined messages. The MAS architecture describes how different agents are interconnecting. MAS are widely used in complex system where abstract definition of certain task is required.

WSNs are considered as complex system where different nodes should cooperate to deliver a task. Moreover, these networks are offering various services that can be independently used in a pervasive way. Basically, WSN will mainly provide monitoring service, notification service, and authentication service. Thus, using MAS in WSN seems very useful. Each of the WSN's service can be viewed as an intelligent agent capable of communication with other agents within the overall MAS. For example, in order to monitor a vital signal of a patient, the monitoring agent is called which, in its turn, calls the authentication agent to verify the access permission of the service requestor (the user who wants to see these vital signals). The implementation of MAS paves the way to reuse some existing agents. For example, a WSN developer can use the authentication agent conceived by Google API[5] or any other authentication agent. Moreover, the power of such architecture is that each agent can be deployed on different nodes, thus enabling task distribution with the WSN. In that way, powerful nodes like server can include agents needing high level of processing, while more constrained nodes like hub or sink can deploy agents for data collection and forwarding. Finally, when defining an agent, you are not only defining a special piece of software written in a defined programming language but also behaviour. Thus, on one hand, you are leaving to the WSN's developer the margin to choose the programming language that suits his specific needs. On the other hand, we are enabling the cross communication between different WSNs where agents can be shared and called when needed. A fully description of our proposed Service oriented multi agent architecture is given in Chapter 4.

1.5 Summary

As we have seen in this Chapter, the use of WSNs is spreading faster and affecting all aspects of our lives. Due to these networks, a wide range of applications are offered such as monitoring, control systems, authentication, online payment, alarms, reminders, etc. However, these systems are very dynamic, especially large scale network where nodes change their state (ON/OFF) and position frequently. Their energy source should be consumed judicially in order to extend the lifetime of the network. In general, the maintenance and reconfiguration of such network is considered expensive and complex. Thus, an effective remote management system will reduce this complexity. The key problem of these networks is the heterogeneity in terms of nodes, data, topology, constraints, applications, etc. This heterogeneity complicates the management and

[5] https://developers.google.com/api-client-library/javascript/features/authentication

design of such networks. We cannot find a common standard that unifies the data management of WSNs. For that reason, the use of modular ontologies facilitates the management of heterogeneity by adding common semantic description to WSN's terms. These ontologies should be part of a multi agent service architecture where sensors are playing the role of service providers and external users/servers are the requestor. The service brokerage is done by querying the ontologies to discover the suitable service. In addition, an independent agent will perform one or more services. These agents will transfer their data using JSON and JSON-LD, and then publish it over the Internet using CoAP or other application protocols. Our proposed ontologies will be defined ad pre-validated in Chapter 2 (Data Model ontology) and Chapter 3 (Service Model Ontology). Our proposed overall multi agent based architecture will be depicted in Chapter 4. This architecture enables the plug and play paradigm, where the sensors and the service are automatically discovered. Furthermore, it expedites the remote management of these networks. This is due to the design of multi agent service oriented architecture that relies on the proposed ontologies. Thus, WSNs now become interoperable, and they can share their data and their service. It is a step toward a standard architecture for data/service management in WSNs. As all people say "Sharing is caring", we believe that if WSNs can share their data and services, advanced caring services can be offered like e.g. wellbeing or lifestyle enhancement.

CHAPTER 2- UNIFIED DATA MODEL FOR WSN "MyOntoSens" Ontology

2.1 Introduction

According to what was already aforementioned in Chapter 1, the following topics are in particular still key stakes for Wireless Sensor Networks WSNs:

- Interoperability management,
- Heterogeneity management,
- Low power,
- Low energy,
- Security.

These stakes are all the more crucial since WSNs usage becomes more and more widespread, including tiny embedded mobile devices and within scenarios of the type Internet of Thing (IoT).

To tackle this heterogeneity, these nodes, as well as the data sensed and/or transmitted, need to be consistently and formally represented and managed through suitable abstraction techniques and generic information models. Therefore, an explicit semantic to every terminology should be assigned and an open data model dedicated for WSNs should be introduced. Thus, our objective is to formalize a semantic open data model for generically describing a WSN, sensors/actuators and their corresponding data. This model should be light enough to respect the low power and therefore low energy limitation of such network, generic for enabling the description of the wide variety of WSNs, and extensible in a way that it can be modified and adapted based on the application.

A deep reading about WSNs topology, architecture, management, and routing, as well as service management and fault management in such networks, was our starting point (Refer to section 1.1). We then explore the sensors/actuators characteristics and components (energy, processor, interfaces, etc.), in addition to the sensed data and features of interest (humidity, temperature…). These readings permitted us to retrieve the domain's vocabulary. Next, we do a survey on the existing models developed for that purposes in order to identify: what already exists and could be reused (partially or totally), and what is really missing and will have to be fully specified in order to bring solutions to WSNs stakes. These researches led us to propose an open data model for WSN and formalize it using OWL language in the form of MyOntoSens ontology. Finally, to pre-validate the proposed ontology, a mobile application for monitoring vital signals of runners during exercises has been implemented fully relying on MyOntoSens ontology.

2.2 State of Art

An overview of existing data representation format and ontologies for WSN will be detailed in the next sub-sections. Moreover, our interest in WBANs let us investigate more deeply the data models dedicated for WBAN and medical EHR systems.

2.2.1 Wireless Sensor Networks (WSNs) common used data representation models and ontologies

In this section, an overview of existing data representation formats and ontologies of WSN sensors/actuators will be presented.

2.2.1.1 OGC SWE Standards

It was proposed by the Open Geospatial Consortium (OGC or OpenGIS) (34) in year 2000 and published as an ISO standard (ISO/DIS 19156). OGC's SWE initiative proposed standards that enable any sensors to be accessible and manageable over the Web. . It provides XML encoding extensions for the description of sensor observations (i.e. sensor data) and their management. The SWE OGC initiative includes the following standards:

- O&M to encode observations and measurements.
- SensorML to describe sensor systems and processes.
- SOS to retrieve and filter observations and sensors' description.
- SPS to request user-driven acquisitions and observations (between a client and a sensor collection system).
- SAS to publish and subscribe alerts.

Focusing on data model for sensor's systems and sensor measurement description, we will only detail the O&M and SensorML standards.

O&M is specified and maintained as a standalone component under the OGC Sensor Web Enablement (SWE) initiative (34). O&M (35) defines the observation as an act associated with a discrete time period through which a number, term or other symbol, is assigned to a phenomenon. The phenomenon is a property of an identifiable object, which is the "*feature of interest"* of the observation. The "*feature of interest*" is a representation of the observation target, being the real-world object regarding which the observation is made. The observation uses a procedure, which is often a device or sensor but may also be a process chain, a human observer, an algorithm, a computation or a simulator. The procedure is used to generate the result. The "*observedProperty"* identifies or describes the phenomenon for which the observation result provides an estimate of its value. Figure 2.1 shows the basic observation model defined in O&M.

An observation has a "*samplingTime"* which is the time that the result applies to the "*feature-of-interest"*. In addition, an observation has a "*resultTime"*, which is the time when the procedure associated with the observation act was valid.

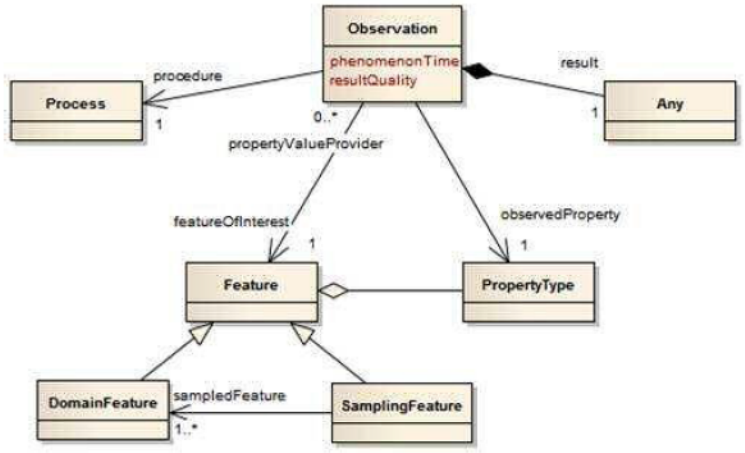

Figure 2.1-Basic O&M Observation Model (37)

An observation parameter is a general event-specific parameter that will be used to record environmental parameters, or event-specific sampling parameters that are not strongly related to either the "*feature-of-interest*" or the procedure. An observation may have both metadata and an indication of the event-specific "*resultQuality"*. Figure 2.2 summarizes the characteristics of an observation. An observation could also have different feature types.

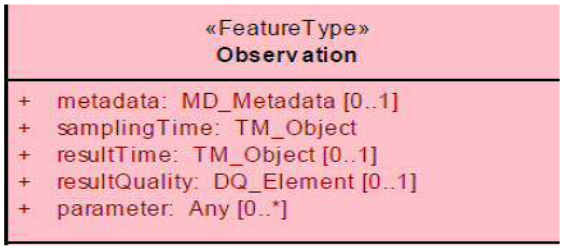

Figure 2.2-O&M Observation Properties (36)

Moreover, some sensors are capable of doing these observations and measurements. That is why OGC defined the SensorML language in order to provide all the attributes required for processing, registering, and assessing the quality of measurements from sensor systems.

SensorML is an XML model and encoding extensions for sensor and sensor data description/processing that has been proposed by the OGC in 2007. It is specified and maintained as a standalone component under the OGC SWE initiative (3). Even if SensorML was initially dedicated for the earth observation domain, it actually serves as basic language for the formalization of many of the existing sensor ontologies. It is described below.

SensorML was defined for handling two possible roles. The first one is to describe the procedure by which an existing observation was obtained. The second one is to provide processing chains from which SensorML-enabled software could derive on-demand new data from existing observations. In SensorML, all components are modelled as processes (36). This includes components, normally viewed as hardware (including transducers and actuators) and processors, sensors and platforms. Processes take input and generate output through the application of an algorithm defined by a method and parameter values. Figure 2.3 depicts the conceptual model of SensorML.

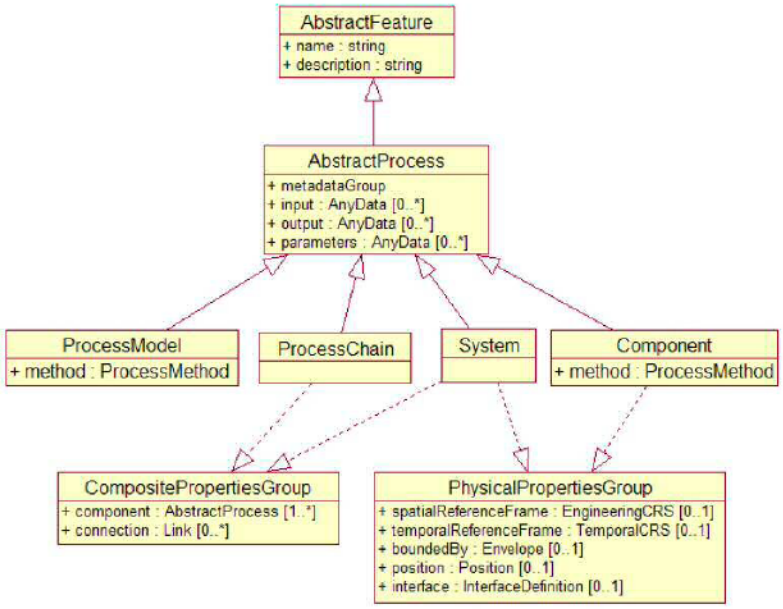

Figure 2.3-SensorML Conceptual Model (38)

Within SensorML, all processes are derived from an "*AbstractProcess"* which is itself derived from an "*AbstractFeature"*. "*ProcessModel"*, presented in Figure 2.3, is used to define more or less atomic pure processes that are expected to be used within more complex process chains. A process chain is defined by the description of all of the processes in the chain and then, the description of the appropriate connections between components. In addition to the description of purely mathematical processes, SensorML also describes the physical processes deriving from the abstract "*PhysicalProcess"* depicted in Figure 2.3. This abstract "*PhysicalProcess"* has "*spatialReferenceFrame"* and "*temporalReferenceFrame"* properties. Those properties define any spatial or temporal Coordinate Reference Systems (CRS) that might be used for interrelating internal components within the process, as well as any reference frame that might allow this component to be related to other components within a process chain or system. "*PhysicalProcess"* may also have zero or more interfaces over which commands and data can flow based on particular physical connections and particular protocols. The interface property contains an "*InterfaceDefinition"* which is based on the Open Systems Interconnection (OSI) reference network layer model, as presented in Figure 2.4.

Within SensorML, any physical process can be modelled as a "*Component"* subdivided into smaller sub processes, or as a single indivisible process. A System is a physical equivalence of a "*ProcessChain"*. A System may include several physical and non-physical processes that all act for providing a certain set of System outputs, based on the System inputs and parameters.

A SensorML resource can have security constraints, valid time and legal constraints which can be a privacy act, intellectual property rights or copyrights. It can take as property a document. Each SensorML system can have a contact which can be a person or a responsible party. Figure 2.5 depicts the properties describing the contact.

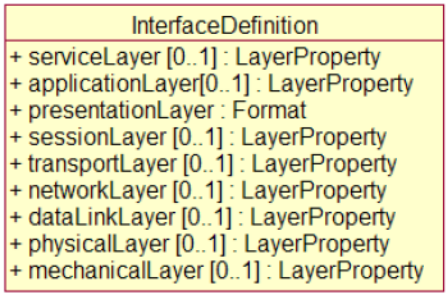

Figure 2.4-SensorML Interface Definition (38)

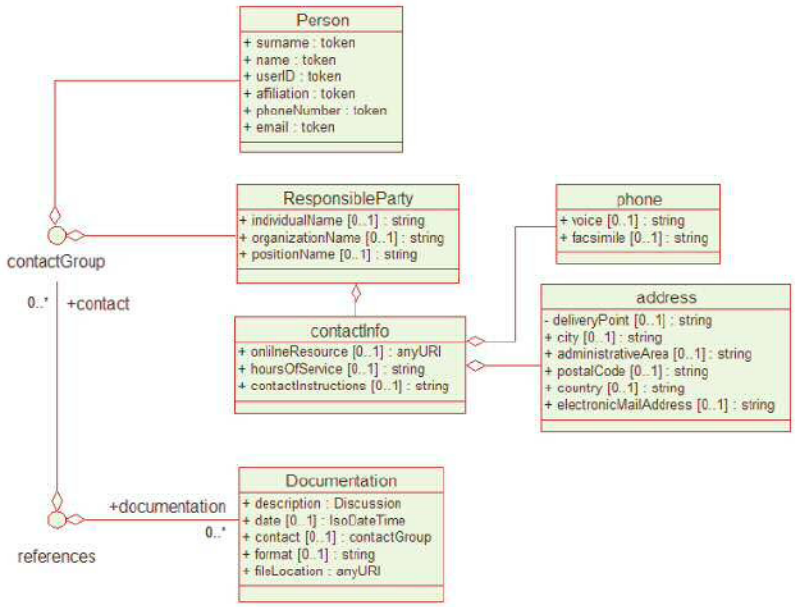

Figure 2.5-SensorML Contact description (38)

2.2.1.2 Existent WSN ontologies

In the last decade, lots of ontologies were proposed for describing WSNs. In (37), the authors state a summary of most important ontologies used. They were themselves inspired by (38) where a detailed analysis of sensor ontologies has been provided, based on two main topics: 'Sensor' and 'Observation'. This analysis is summarized in the Table 2.1.

Table 2.1 shows that while Avancha ontology focuses on the observation's features, it ignores the sensor characteristics. Moreover, CSIRO ontology seems to be the ontology which fully describes the sensor and its physical features. However it doesn't describe the data transferred between sensors.

Advanced ontologies have been investigated introducing new description's features representing the WSN's data.

2.2.1.3 OntoSensor Ontology

OntoSensor (4) is stemming from a combination of SensorML defined by OGC, IEEE Suggested Upper Merged Ontology (SUMO) and the International Organization for Standardization (ISO) 19115. Its latest updated was released in 2008. The main objective of OntoSensor is to provide an ontological description for data observation of sensors. Figure 2.6 shows the different parts of OntoSensor Ontology.

OntoSensor ontology supports data discovery, processing and analysis of sensor measurements, geo-location of observed values (measured data), performance characteristics (e.g. accuracy, threshold, etc.), and explicit description of the process by which an observation was obtained (i.e. its lineage). It also provides an executable process chain for deriving new data products on demand.

Although OntoSensor illustrates a semantic approach to sensor description and provides an extensive knowledge model, there is no distinctive data description model to facilitate interoperable data representation for sensors observation and data. This is however quite mandatory for WSN's description.

Table 2.1- Comparison between existing ontologies based on Sensor & Observation

	Sensor								Observation				
Ontology	*Physical Characteristics*	*Hierarchy*	*Contacting & software*	*Identity & Manufacturing*	*Components & process*	*Configuration*	*history*	*deployment*	*Field of view\sensing*	*accuracy*	*frequency*	*Response model*	*Unit of measurement*
MIMI	x			x	x	x		x		x	x	x	
OOSTethys	x				x				x	x	x	x	
Kim						x			x	x	x		
Eid	x			x		x			x	x	x	x	
Matheus	x	x	x		x						x		
Avancha	x		x						x	x	x	x	x

2.2.1.4 Semantic Sensor Network ontology (SSN)

The most important ontology that includes sensor, observation and data description is Semantic Sensor Network ontology (SSN) (4). Developed by W3C Semantic Sensor Network Incubator Group, this ontology describes the capabilities of sensors, the measurement processes used and the resultant observations. SSN can also be aligned with other ontologies that describe e.g. observed phenomena. This ontology covers large parts of the SensorML and O&M standards from the OGC, omitting calibrations as well as process descriptions and data types, which were deemed not sensor specific (if required it can always be included from other ontologies). However, SSN does not provide facilities for abstraction, categorization, and reasoning offered by semantic technologies.

Figure 2.7 shows that SSN is focusing on the deployment of the wireless sensor network, the description of the sensors and the phenomena it is measuring. The ontology can be used for a focus on any (or a combination) of a number of perspectives:

- A sensor perspective, with a focus on what is sensed, and how it is sensed,
- A data or observation perspective, with a focus on observations and related metadata;

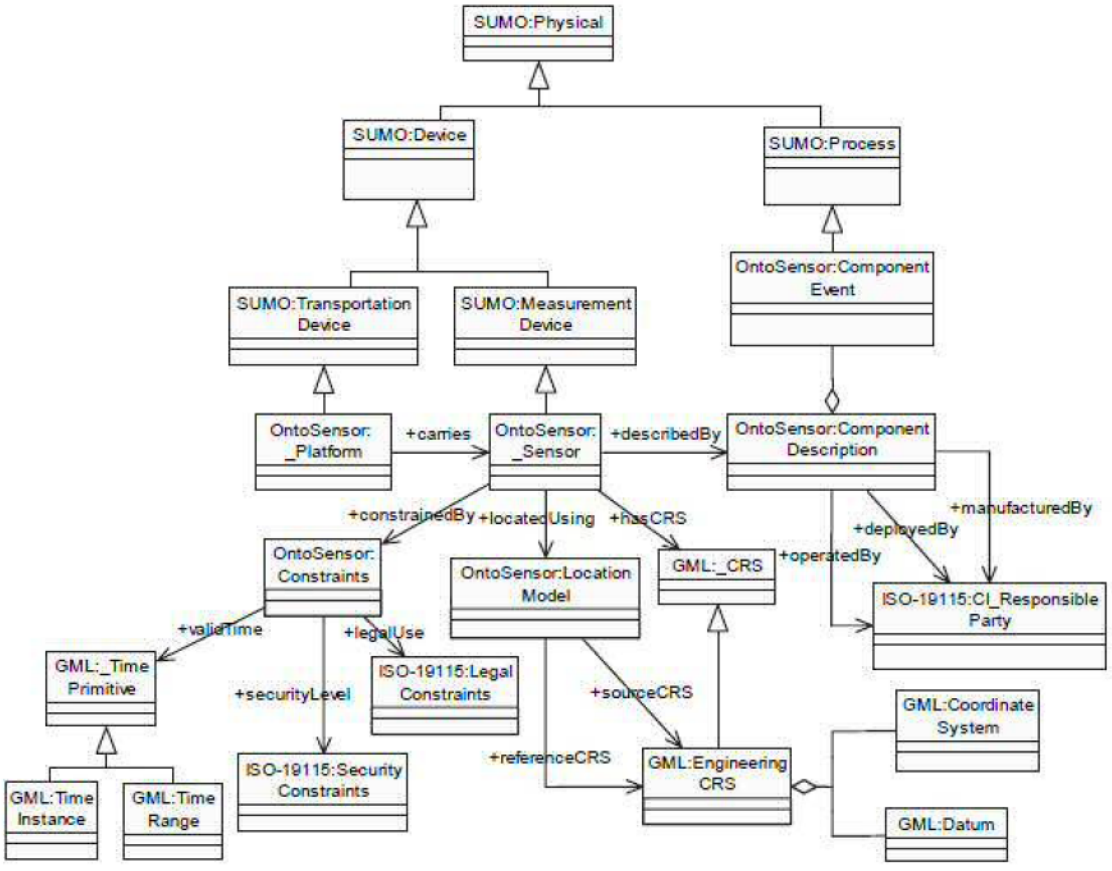

Figure 2.6-OntoSensor Ontology (4)

- A system perspective, with a focus on systems of sensors,

- Or a feature and property perspective, with a focus on features, their properties, and what can sense those properties.

However, SSN ontology does not focus on the communication process. In order to overcome this limitation, W3C members introduced, in the eleventh International Semantic Web Conference, an extension of SSN ontology called Wireless Semantic Sensor Network ontology (WSSN).

2.2.1.5 WSSN Ontology

WSSN (39) was proposed by W3C to extend SNN with three new concepts: the communication process, the data stream and the state. The "Communicating class" is added to the ontology for describing the sensor communication process. In addition to that, a "*CommunicationOutPut*" class is added for describing data transmitted by the sensor (the communicating device). The resulting data stream, if a communication occurs, is defined as a set of communications while the acquisition data stream is defined as a set of observations. A communicated data can be equal to a data acquired by sensor or to a data generated from a set of data acquired by sensors using, for example, some aggregation procedure. The WSN node and the observed phenomenon pass through several states. In the WSSN ontology, the state of any entity is represented by the "State class".

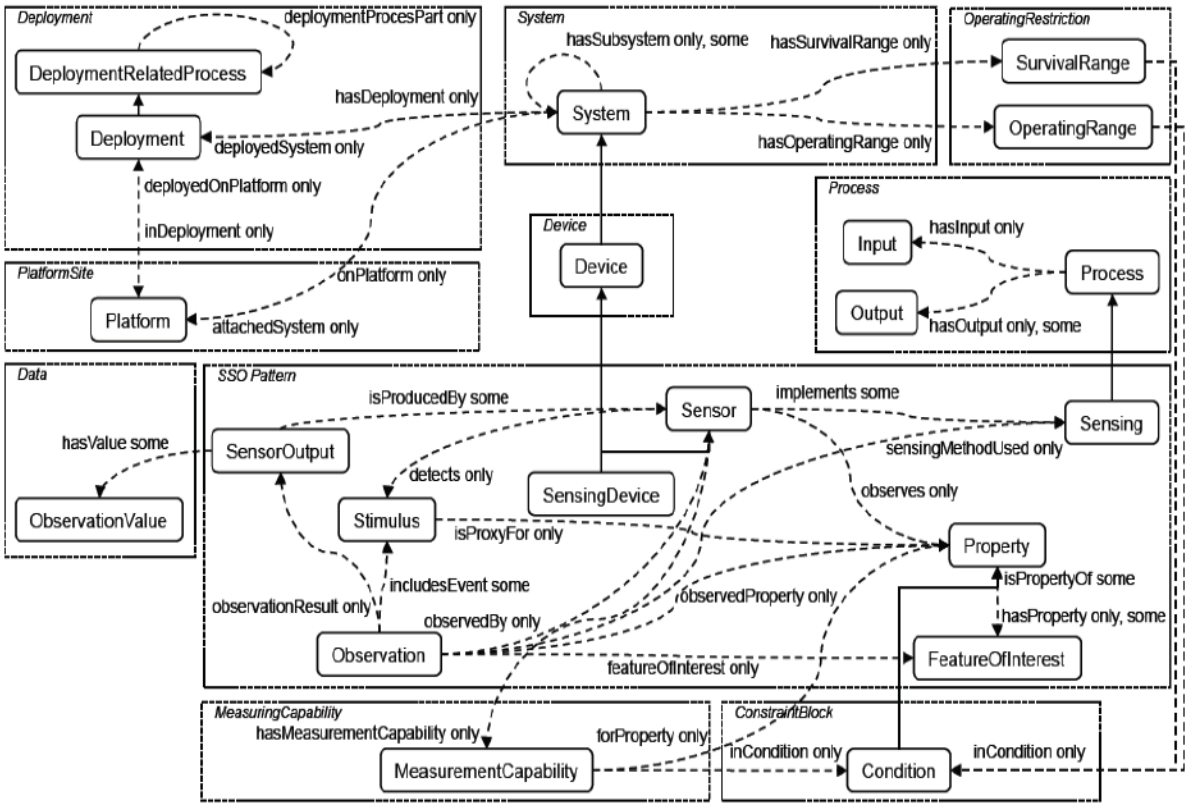

Figure 2.7-SSN Ontology (3)

2.2.1.6 Semantic Web Based Architecture for Managing Hardware Heterogeneity (40)

A new semantic ontology was proposed in by Nikolić *et al*. (40) in 2011 for managing hardware heterogeneity in wireless sensor network. The proposed architecture, "Semantic Web Based

Architecture for Managing Hardware Heterogeneity", consists of three segments: Application, Middleware and WSN. Figure 2.8 shows that this ontology describes the sensor, its physical features (dimension, platform.), the connection and the data measured. It uses the same features described in SensorML to describe a node property, a component and the data. Moreover, it includes the following additional features:

- Data & Data Stream. The ontology uses "*DataTime*" and Value fields in the "*ReadingSensorstoTable*" to describe the data stream,
- Communication Process & Policy. The ontology includes Radio component that describes the "*RadioType*" and "*RadioFrequency*",
- Acquisition Policy. This field indicates the interval of data acquisition,
- Data Quality. However, this field is not covered by the ontology.

Moreover, "Semantic Web Based Architecture for Managing Hardware Heterogeneity" ontology includes a Processing Component composed of Memory Component, not mentioned in the ontologies cited previously. Figure 2.9 shows the Component structure in this ontology. The Processing component may be very useful for the management of the Wireless sensor Network because it reflects the capability of a node in terms of energy source and memory structure.

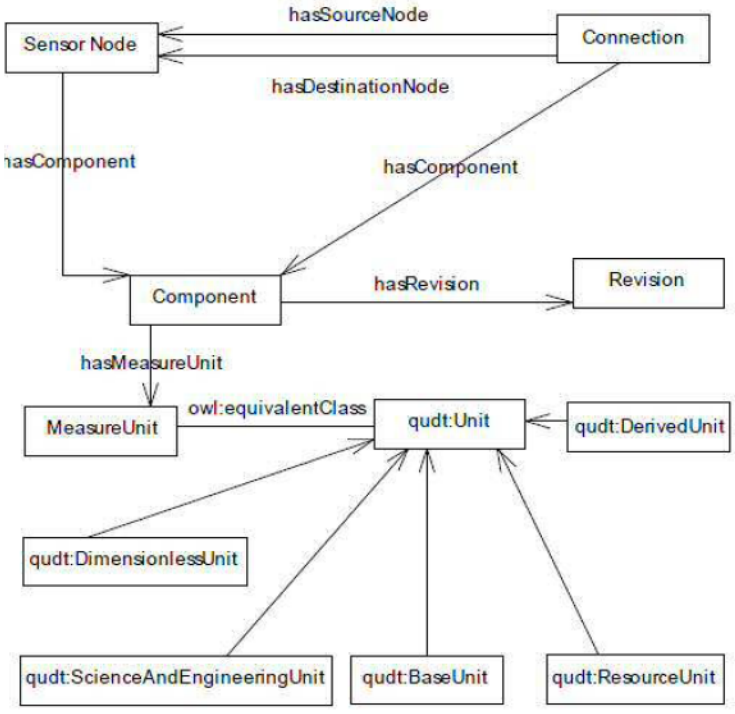

Figure 2.8-Semantic Web Based Architecture for Managing Hardware Heterogeneity Ontology (41)

In 2013, the National Institute of Standard and Technology (NIST) proposed the requirements for designing the manufacturing perception sensor ontology (41). The list of these required attributes is given in Table 2.2. NIST mainly introduced new attributes that describe data type, state, confidence and provenance. Moreover, NIST has also introduced network

topology/geometry and resource processing in their ontology. However, NIST ontology is still under construction.

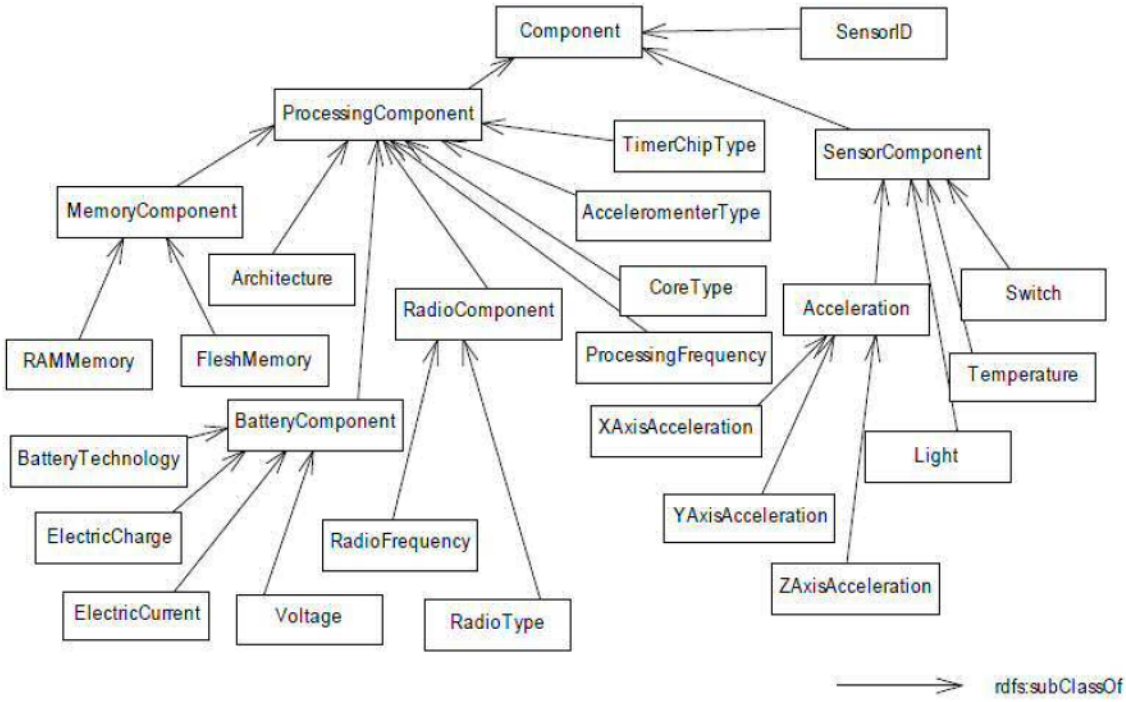

Figure 2.9-Component Class Structure in Semantic Web Based Architecture for Managing Hardware Heterogeneity Ontology (41)

Table 2.3 gives a comparison between the existing ontologies for WSN based on three main axes Sensor, Observation and Data

In conclusion, in spite of its quite extensive semantic knowledge model, OntoSensor does not incorporate any distinctive data description model for facilitating observation and data representation interoperability. This is however mandatory for WSNs' data management. Moreover, The CISRO ontology, presented by M. Compton in 2009, focuses on describing and reasoning about sensors and observed process, without covering the data part. The Semantic Sensor Network Ontology (SSN) describes the capabilities of sensors, the measurement processes used and the resultant observations. SSN can also be aligned with other ontologies that describe, for example, the observed phenomena. This ontology covers large parts of the SensorML and O&M (Observations & Measurements) standards from the OGC, omitting calibrations as well as process descriptions and data types which were deemed not sensor specific (if required, it can always be included from other ontologies). However, SSN does not provide facilities for abstraction, categorization, and reasoning offered by semantic technologies. Furthermore, Semantic Wireless Sensor Network Ontology (SWSN) extends SNN with three new concepts: the communication process, the data stream and the state. The SWSN ontology includes Sensor, Observation and Data descriptions. Unlike previous ontologies, this ontology includes an additional processing component composed of the memory component. However, SWSN does not include security and QoS features that are needed for WSNs' fault tolerance

management. Nevertheless, to efficiently manage a Wireless Sensor Network, more components than the ones already proposed need to be added for conflict resolution, similarity detection, data fusion, routing, error detection, etc. Let us precise that our proposed ontology will overcome these limitations by:

- Including all the sensor, observation and data features,
- Introducing new components related to WSN management (power, traffic and fault management), the state of links and nodes, the security features and the Quality of Service (QoS).

Table 2.2-Requested attributes for WSN description proposed by NIST

Attributes	**Comments**
Physical	Power, weight, size
Operating Conditions	Environmental conditions required for operation
Immediate Data	Characteristics of data, resolution
Derived data	Results computed from raw data
Algorithms	Alternative algorithms for producing derived data
Integration/fusion	data combined from different sensors
Capabilities	Functional applications of raw and derived data
Communication	Physical and logical protocols
Processing	On board processing power of sensors and network nodes
Calibration	Individual and joint sensor calibration information
Provenance	Record of sensor and processing history
Confidence	Level of confidence in derived data

In summary, none of the existing ontologies are fully providing a solution to heterogeneity and interoperability management for WSNs since they are only covering a part of the required information to model. Furthermore, no combination of the existing ontologies is able to provide a complete level of reasoning over sensor data and measurements since the resulting model will anyway lack extended semantic for that purpose.

2.2.2 Existing E-health data models

CEN/TC251 has been founded in 1990 (42) (43). Its objective is the standardization in the field of health Information and Communications Technology (ICT) to achieve compatibility and interoperability between independent systems and to enable modularity. CEN/TC251 has developed EN13606 for Electronic health record communication which follows a dual model

methodology. The CEN/TC251 EN13606 norm is composed by five parts that are summarized below.

Table 2.3-Comparison of existing ontologies based on Sensor, Observation & Data

	Sensor				Observation							Data					
Ontology	*Physical Characteristics*	*Hierarchy*	*Deployment &*	*Identity & Manufacturing*	*Inputs\Outputs*	*Field of view\sensing*	*accuracy*	*frequency*	*Response model*	*Unit of measurement*	*Communication Process*	*Data*	*Sensing process*	*Data Type*	*Data Quality*	*Acquisition Policy*	*Data States*
CESN	x	x	x									x	x			x	
CSIRO	x	x	x	x			x	x	x	x		x	x			x	
Sensi O&M												x	x			x	
OntoSensor	x	×			x	x	x	x	x	x		x	x			x	
SSN-XG	x	x	x	x	x	x	x	x	x		x					x	
WSSN	x	x	x	x	x	x	x	x	x		x		x			x	
SWSN	x	x	x	x	x	x		x	x	x	x	x	x	x		x	

Reference Model

The Reference Model depicted in Figure 2.8 formalizes the global characteristics of health record components, i.e. how they are aggregated, and the context information required meeting ethical, legal and provenance requirements. Entry, Composition (which is the set of information committed to one Electronic Health Record - EHR - result) and elements of each EHR describe the clinical data related to the patient, the diseases and the collected data. The cluster allows the specification of nested multi-part data structures such as time series, list or Tables.

Each element has one "*Data_Value*". The "*Data_Value*" summarized in Figure 2.9 of the reference model describes all the types of data that can be found in the biomedical environment.

Archetype Model

An archetype is expressed in the form of constraints on the Reference Model. These constraints can consist in defining relevant attribute values, optionality and multiplicity of attributes and objects, data types and value ranges that data instances may take, and finally binding to clinical terminologies or ontologies. The archetype model will group the data collected from the ontology to respect the adequate format and repartition.

Reference Archetypes and term lists

The term lists contain a list of biomedical terms. They can be used as data dictionary for the service layer.

Security

Health records have to be created, processed and managed in ways that guarantee the confidentiality of their contents and legitimate control by a patient in their usage.

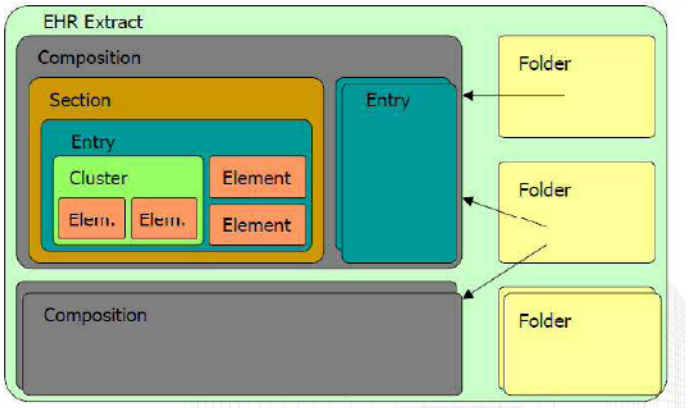

Figure 2.10-CEN TC 251 Reference Model (44)

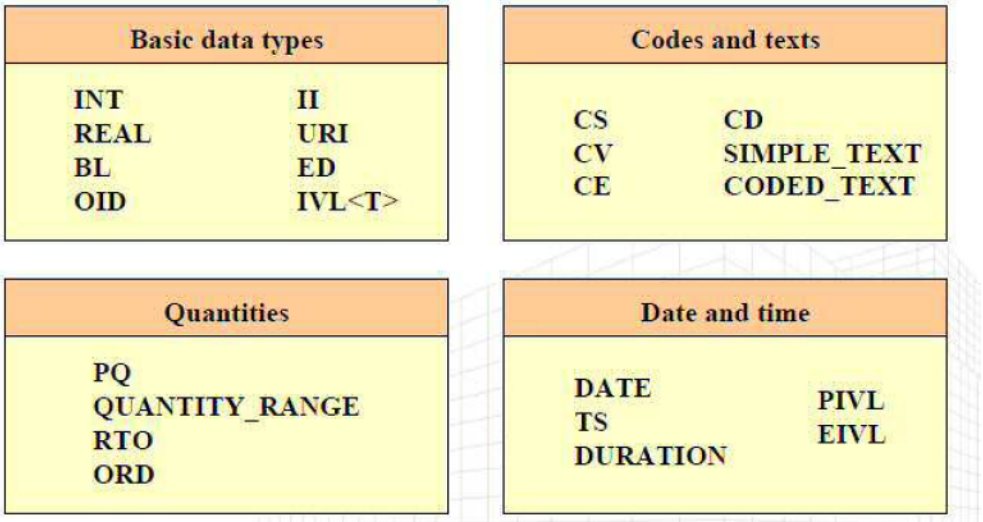

Figure 2.11-Data_Value in Reference Model (44)

Interface Specification

The interface specification defines the computational view point of each interface as the payload to be communicated.

IEEE reference models for medical device communications (ISO/IEEE 11073 Personal Health Device (PHD) (44)) standards are a group of standards addressing the interoperability of personal health devices (PHDs) such as weighing scales, blood pressure monitors, blood glucose monitors, ... The standards draw upon earlier IEEE11073 standard works, but differ from those earlier works due to an emphasis on devices for personal use (rather than hospital use) and the use of a simplified communication model. The central core of the standard is the so-called Domain Information Model. Objects containing vital-sign data representations and their relationships are defined in this model. Objects for additional services around vital signs data, are also defined. The ISO/IEEE 11073 model is depicted in Figure 2.12.

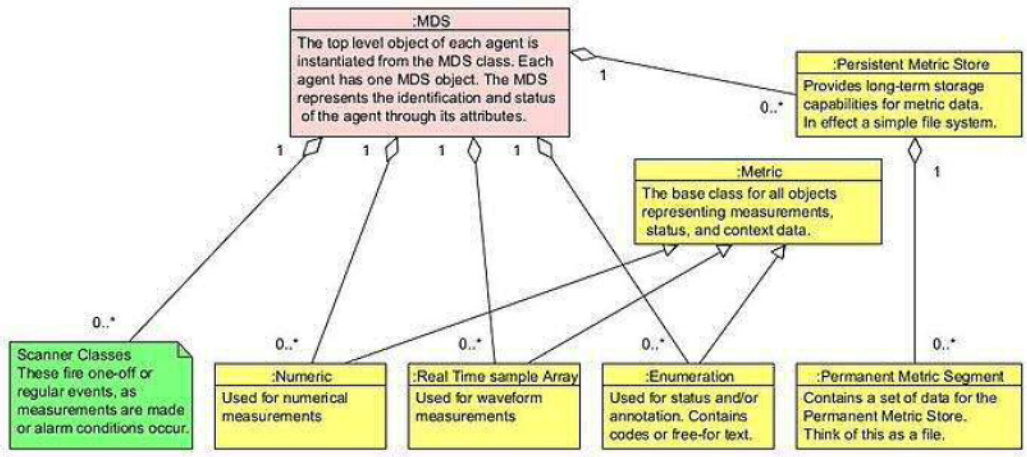

Figure 2.12-ISO/IEEE 11073 Model Summary (46)

Numeric objects relate to the physiological parameters. They have as attributes: the mechanism to obtain an observed value and its status such as units and the timestamp. The Real Time and Enumeration attributes are used to represent a group of data. The Scanner Class determines the way the data is transmitted from sender to destination. It provides mechanisms to reduce message overhead by communicating the message format at configuration time, and then sending pure data during run time. The Storage Class, called PM-Store, provides a facility to store data for longer periods of time and also provides mechanisms to transfer portions of data that are needed by the receiver. Within the IEEE 1073 standard, device specialization sub-standards have been described.

Bluetooth LE (Low Energy) (45) was proposed and specified by the Bluetooth SIG for providing the current standard with lower energy working process, thus allowing its usage in button cells and energy harvesting applications (like e.g. e-Health or Internet of Things applications). For that purpose, Bluetooth SIG (Special Interest Group) members fully redesigned the physical and MAC layer of Bluetooth, as well as its protocol stack. At the upper layer level, they introduced a simplified device and application profile called GATT (Generic Attribute Profile), provided with the corresponding discovery and access protocol called Attribute Protocol. All the profiles that will be introduced within Bluetooth LE will be derived from the generic GATT one.

Within GATT, all the data are exposed as attributes. An attribute has:

- A unique identifier called UUID,
- A type,
- A value, with fix or variable length (from 0 to 512o),
- And a permission. The permission is used for setting the attribute's access rights for reading and or writing.
- A device is characterize by the following attributes:
- Manufacturer Name, a String representing the name of the manufacturer of the device,
- Model Number, a String representing the model number that is assigned by the device vendor,
- Serial Number, a String representing the serial number of a particular instance of the device,
- Hardware Revision, a String representing the hardware revision for the hardware within the device,
- Firmware Revision, a String representing the firmware revision for the firmware within the device,
- Software Revision, a String representing the software revision for the software within the device,
- System ID, a structure containing an Organizationally Unique Identifier (OUI) followed by a manufacturer-defined identifier. This ID is unique for each individual instance of the product,
- IEEE 11073-20601 Regulatory Certification Data List, representing regulatory and certification information for the product in a list defined within IEEE 11073- 20601.

Within the GATT specification, units are given using the international standards for the measurement of physical quantities, such as e.g. Meter for length or Kilogram for mass. The type of the units is determined in the "*Format Types*" attribute.

The "*Attribute Protocol*" is a client/server based protocol providing attribute exchange functionalities between devices. GATT and "*Application Protocol*" upper layer modules are also associated with "*Generic Access Profile*" (GAP) and "*Access Manager*" modules that provide secure device discovery, access and connection functionalities. The corresponding device discovery, access and connection profiles are specified within the GAP module.

Based on that new standard, the Continua alliance specified new device/service profiles dedicated to medical devices and e-Health applications. It defines 13 different profiles that determine how services can be used to enable an application or use case. Some of these profiles are dedicated for notification, while some others are dedicated for measurement and for client server communication. These profiles are detailed below.

The HID Over GATT Profile (HOGP) defines the way a Bluetooth LE device can support Human Interface Device (HID ; e.g. Heart Rate Monitor, Glucose Meter, Changed Oximeter, Mobile Phone) services using the Generic Attribute Profile.

The Measurement Profiles are used to determine how a Connector could connect and interact with the sensor for being used in health care applications. It includes blood pressure, cycling power, cycling speed and cadence, glucose, temperature, heart rate, location and navigation, proximity, running speed and cadence. The Time profile is used to get the date and time attributes.

The Alert Notification Profile (ANP) enables a client device to receive different types of alerts and event information. For example, the Phone Alert Status profile is used for obtaining the phone alert status exposed by a phone. The Scan Parameters Profile (SPP) is used for providing devices with information assisting them in the management of their connection idle timeout and advertising parameters in order to optimize their power consumption and/or reconnection latency.

E31 Committee on Healthcare Informatics of American Standards for Testing and Materials (ASTM) (46) develops standards related to architecture, content, storage, security, confidentiality, functionality, communication of healthcare information and decision making (including patient-specific information and knowledge). ASTM E31 has also developed some standards for defining health vocabularies, Electronic Health Record (EHR) system management (including interactions between different parties), as well as for ensuring security and privacy of Healthcare systems.

Four messaging standards were created:

- ASTM E1238-97 standard specification for transferring clinical observations between independent computer systems (Withdrawn 2002),
- ASTM1394 standard specification for coded values used within the EHR. This standard allows the representation of different biomedical signals, with the cooperation of Health Industry Level 7 (HL7), for supporting multichannel signals,
- ASTM E1467-94 (2000) standard specification for transferring digital neurophysiological data between independent computer systems (Withdrawn 2004). It defines the message structure exchanged between clinical instruments.
- E2369 standard specification for the Continuity of Care Record (CCR). This recent specification gives a particular and detailed description of data (and its organization into structured textual forms) that are dedicated for the communication between healthcare enterprises and that may not be interoperable otherwise. This specification details the abstraction of data contained in the EHR attributes. These data are documented in practice E-1384 and are organized and composed into a textual view for their transfer to other care settings that may not utilize a structured electronic representation of patient data. The textual form is used in order to provide a familiar cognitive format for information exchange between practitioners that are currently operating using traditional unstructured care settings.

In summary, on the e-Health side, lots of standards for healthcare message exchange and terminology have been developed such as CEN TC251, IEEE 1173, ASTM E31 or Bluetooth profiles for medical devices. CEN TC 251 (EN13606 standard) defines the high-level logical models for any kind of EHR and hereby enables syntactic interoperability. However, this only guaranties common data structure exchange procedures but does not ensure that the data meaning will be interpreted identically by all parties. Regarding IEEE 11073, it defines the data format, in addition to the data storage and data transmission, for e-Health environments. ASTME31 short analysis shows that this standard is the closest to an EHR object model. However, it is restricted to laboratories instruments description, medical knowledge representation, digital neurophysiological data, and clinical data transmission. Unfortunately, ASTME31 is not so extensible and still lacks a lot of essential information like prescription information. Finally, Bluetooth LE alleviates the interoperability between medical devices by presenting specific data formats and units, in addition to the notification systems and communication between different medical parties. Unfortunately it is only dedicated for Bluetooth enabled devices.

Thus, none of the existing ontologies, as well as no combination of the existing ontologies, is able to provide a complete level of reasoning over WSN sensed/medical data stored in multiple locations. That's confirms the need of a specification and qualification of a unified extended data representation model and transfer format in order to enhance the management of WSN systems and integrate it in WoT. For that purpose we proposed MyOntoSens ontology, detailed in the next sections.

2.3 Proposed Unified Data Model for WSN

Dealing with tiny devices in WSN pushes us to create a modular model where some classes could be implemented in the sensor/actuator nodes depending on its resource's availability, and some other could be implemented and processed in more capable nodes like e.g. the Hub (in other terns sink or coordinator). Basically, the model is divided into three main parts: WSN, Nodes (i.e. Hub, sensors, and actuators), Process and Measurements. Each of these parts is detailed in the following sub-sections.

2.3.1 WSN Data Model

First, a WSN is identified by its WSNID that should be unique and accessible by any authorized user. Second, the WSN's location is essential in network discovery as well as in network cooperation. For example, if a WSN is monitoring the temperature of a home to protect it from fire, it will be very useful to propagate fire detection along other neighbours' home system. Thus, the location of WSN is taken into consideration in our WSN data model. The location could be an address or GPS coordinates (like it is described in SensorML xml schemas).

The WSN will monitor a specific phenomenon (burned calories during exercises, glucose level, fire detection, traffic control, etc.) belonging to a specific domain of application (healthcare, telemedicine, assisted living, sport training, vehicle networks, smart grids, etc.). These two properties are used in our proposed WSN data model to retrieve, when needed, the WSNs monitoring the same phenomena in order to enhance WSN's cooperation and do certain data analysis. Let's consider a building where each apartment is equipped with a WSN dedicated to monitor the quality of air. It is obvious that if these WSNs communicate their information, it will

be easier for environmental expert to identify the degree of pollution in certain region and propose more solution to enhance the user's quality of life.

The required lifetime and the fault tolerance differ from one application to another. Whereas WSN for entertainment purposes should have a lifetime of weeks or few years, WSN dedicated for assisted living or anomaly monitoring could last for many years. In addition a WSN should measure accurate value and requires a small fault tolerance. This parameter is essential in the management of the networks where the fault tolerance should be respected. It may imply nodes reconfiguration where some nodes are considered down due to the inaccurate sensed data.

A WSN is formed of clusters or directly contains nodes. Theses nodes are distributed based on a physical topology. Each cluster is identified by a cluster ID and described by a channel and frequency. The data within the WSN or cluster are exchanged using certain radio technology like e.g. ZigBee, Bluetooth Low Energy, UWB (Ultra Wide Band) and others. If Bluetooth is used for wireless communication, the channel for each cluster is saved in addition to the number of nodes and the topology used within the cluster.

It is essential to keep track of the contact person and/or the responsible party of the WSN. We re-utilized the contact model described by SensorML where a contact can be a person described by his name, *UserID*, email and phone number. A contact can also be an organization described by its name and address. In our model, a contact can be:

- An online resource or document. In that case, a reference to the document is simply given as the class attribute.
- A Responsible Party which plays the role of the patient's reference. It can be an organization or a person. The creation of the responsible party is very useful for emergency cases. For example, if the WSN is developed for the monitoring of elderly, in urgent cases, the system can generate a notification and send it to the responsible party's mobile phone.
- A person which can be a Patient. The Continua Alliance describes the patient information in the user data service within the Bluetooth LE standard by patient's full name, weight, height, email, date of birth, gender, resting heart rate, waist circumference, hip circumference, fat burn heart rate lower limit, fat burn heart rate upper limit and language.

In order to enhance the management of electronic health data shared between medical systems, we considered the properties defined by Continua Alliance and linked the diseases, blood type, allergy or any other medical descriptions properties inside our model. The patient can also be identified by his user ID or social security number. Sharing these data among different biomedical systems allow any clinic or hospital to identify the patient and deduce his medical archive.

In addition, a WSN can have an owner. This relation is in particular useful in WBAN monitoring systems where the owner will be the monitored patient and the contacts will be the medical's staff and patient's relatives. Because a WSN can be composed of many clusters, the relation *uses* has been added between the patient and the cluster.

In WSN architecture, we can differentiate between three types of communication, Intra WSN communication between the nodes and the Hub or sink, Inter WSN communication between the Hub and the internet and beyond WSN communication with the monitoring/control servers and domain experts servers on the cloud. The inter WSN communication will obey to one of these three ways. The first way is the periodic communication where the nodes send data to the WSN's coordinator (i.e. the cluster Hub) every T seconds. The second one is based on WSN's coordinator requesting WSN's nodes for triggering their reply. The third one is based on event-triggering. When certain events occur, the nodes send their data to the WSN's coordinator. That is why we are adding the inter WSN communication type in our WSN data model to enhance remote management. For example, if periodic exchange is used and the hub did not receive data during T seconds, it can identify that this node is malfunctioning. Figure 2.13 depicts the WSN Data Model.

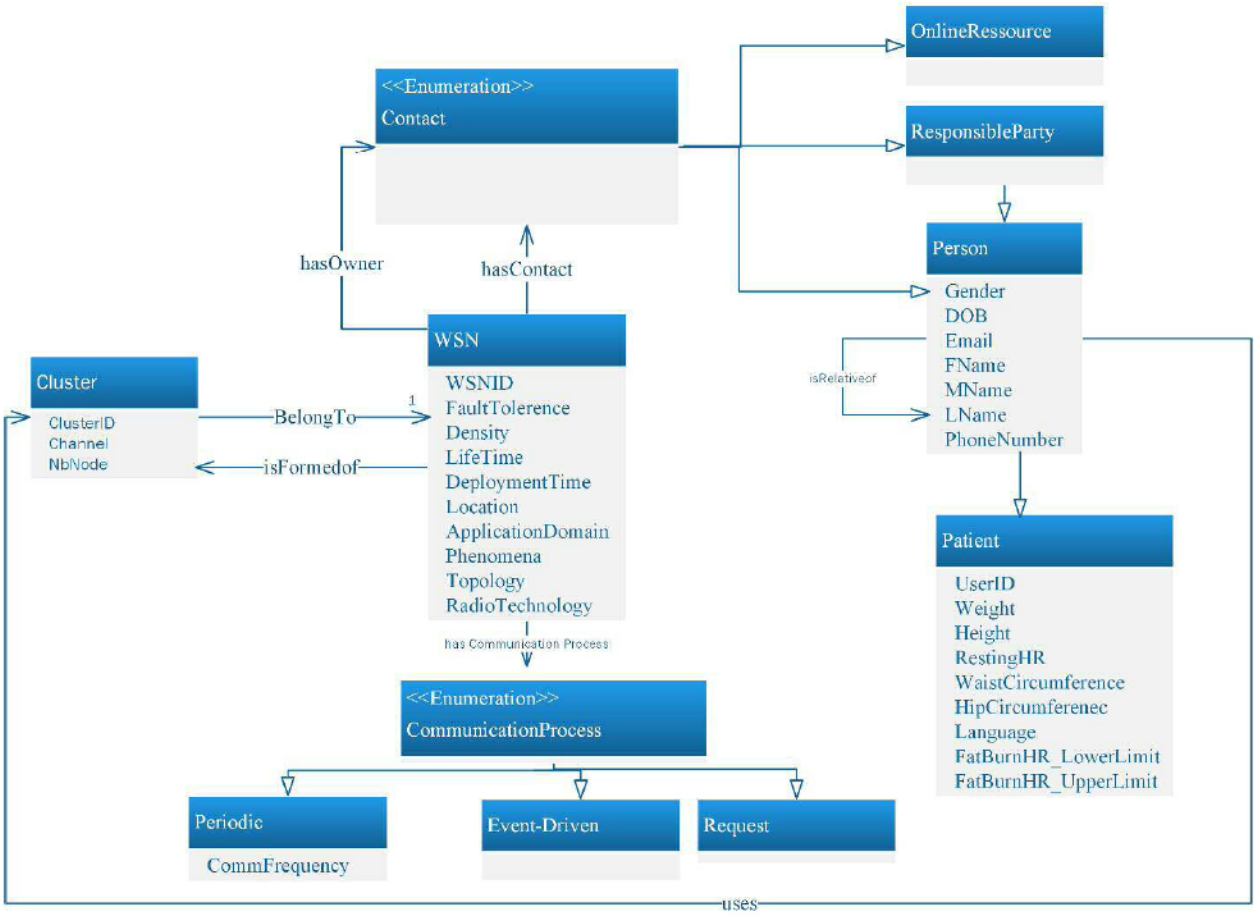

Figure 2.13-WSN Data Model

2.3.2 Nodes Data Model

The node can be a sensor, an actuator which acts according to data received from the sensors or through interaction with user, or a node responsible of data routing or pre-processing like a WSN coordinator or relay, a cluster head (WSN's hub), or others. As an example, an actuator equipped with a built-in reservoir and pump can administer the correct dose of insulin to give to diabetics based on the glucose level measurements. The node can be a cluster head (WSN's Hub)

which is responsible for collecting the data and for ending it outside the WSN to a local or remote monitoring and control centre. In some cases, the coordinator is called aggregator, collector or body centre unit. In our model, the coordinator is called the Hub. In many cases especially in m-health applications, the SmartPhone or the PDA of the user will play this role. It will collect the data from the sensors via Bluetooth interfaces and send it to the medical servers via 3G/4G or wireless interfaces. Thus, in our Nodes model, the node is a class having three subclasses: Hub, Sensor, and Actuator.

Moreover, each node is identified by the Node ID that can be the MAC address, the serial number (as defined in SensorML schema), an URI, or any other unique descriptor. To avoid redundancy, we have created a new class named *NodeType* that describes the physical characteristics of a node. Each node has only one node type. We have added the survival properties (mentioned in SSN ontology) which describe the conditions to which a sensor can be exposed without causing lasting damage. When these properties occurred on a network, we should consider all data sent from this node as insignificant data and trigger the stopping of this node.

The mobility also enhances the performance of the sensor networks. Basically, the nodes in a WBAN have mobility capability. Thus, the velocity of a mobile sensor (as described in SensorML schema), the energy consumption during its movement and the maximum distance that it can travel should be registered in our model. This feature will help in collecting data. Many times, it will be more efficient that a mobile sensor traveling over the network collects data or sends requests or reconfigures certain nodes.

Each node type has certain functioning modes (Sleep, active, down, transmission, reception, etc.). Although WSSN ontology introduced the concept of state (same as mode in our proposed model), it didn't add the power consumed in each mode. This is critical information for energy efficiency management of WSN, especially when the sensors are auto-powered with low capacity cell batteries. Thus, keeping in mind the WSN's low energy constraint, we have added the power consumed in each mode to our model in order to estimate the lifetime of the WSN. For example, if the communication within the WSN is a periodic communication, it will be useful to switch the node from active mode to sleep mode during a period t (since no data will be sent during that period), thus permitting a reduction of power consumption on the node.

Each node type has wakeup latency. The wakeup latency is the time required by the sensor to generate a correct value once activated. Clearly, if the sensor reading is performed before the wakeup latency has elapsed; the acquired data are not valid.

Furthermore, each node type has a processor identified by a processor ID. The processor has RAM, ROM and a flash memory. We reused the idea of Memory components defined by Nikolić *et al.* (40), but we separated the information into two classes: *MemoEnergy* (for current node's memory capacity) and *Processor* (for static nodes' memory capacity). By introducing the *MemoEnergy* class, the node with higher memory resources can be prioritized to be elected as a cluster head or a WSN's coordinator. In addition, we introduce the MIPS, frequency and duty cycle to our *Processor* class model because these attributes reflect the speed of processing certain data.

While the nodes are typically powered by batteries with a limited lifetime, they can benefit from additional energy harvested from the external environment, Nikolić *et al.* (40) have introduced

in their model the Battery Components class. However, many researches have been conducted in the domain of energy scavenging from on-body sources, such as body heat and body vibration. Thus, because the source of energy for the same node can vary, we insisted on revealing the source of energy in our model, in the *EnergySource* class. The property *Rechargeable* has also been added to be aware that if the source is un-rechargeable, the lifetime of the WSN will be affected by the available battery level of the node. Otherwise, a notification can be sent to the owner or responsible of the WSN to recharge the battery of the node.

Data transmission within a WSN is done via interfaces. Each node type can have many interfaces (e.g. Bluetooth, UWB, IEEE 802.15.6, serial,). To avoid redundancy, we created a new class *Interface Type* that describes the standard characteristics of the interface's protocol, the baud rate and the functional layer (influenced from SensorML interface schema). Regarding the interface, it is identified by the interface ID that can be the IP address or the unique address used in the communication protocol. The interface can have many addresses like MAC address, IP address or others. We can also precise if the interface is considered as gateway in order to help the monitoring and control server in finding how it can communicate with the WSN from its LAN, backbone, or infrastructure side. All these interfaces' properties have been added to our *Interface* and *Interface Type* classes.

We have seen the static description of the nodes, now we present the dynamic characteristics that vary during the WSN lifetime. First, each node has only one current mode. For example, if a WSN's coordinator has sent a request to collect certain information, only nodes who are in Active mode can reply. Moreover, the patient's information should be always available to the physician. By tracking the current mode of each node, in addition to the features mentioned before, we can choose how the switching process can be done. The node occupies certain position. The position attribute can be:

- described by coordinates using a reference system, like it is described in SensorML, or
- what we have also added within our model, planted in the human body, on the body surface (head, waist, chest, wrist, etc.), or not in contact with the human body (with a distance of few meters).

Moreover, with the growth of m-health applications, the position could be built-in the mobile phone.

Recently, many researches focused on hop to hop routing mechanism in WSN. Thus, to enhance the routing management, the *level* property has been added in our model to the *Node* class. This property indicates the distance (as hop count) from the node to the sink or hub. Moreover, security is a key point in WSN's management and that is why we added the *Trust Level* for each node within or model.

We also focused on the dynamic memory and energy components of the nodes that are described in the *MemoEnergy* class, like previously mentioned above. The remaining lifetime or memory can directly influence the selection of the WSN's Hub or coordinator, especially because the hub's death can cause the malfunctioning of the entire WSN.

Finally, a crucial issue is the reliability of the transmission in order to guarantee that the monitored data is received correctly and in reasonable time. In general, the Hub is the node that can monitor other nodes to predict any failure. To assure the reliability, we proposed to add the

link state for the path between nodes for indicating the last time the Hub has received a message from a node. If this time is too long, the Hub can therefore predict that this node is facing some trouble. In addition, it will keep track of the *error rate* of a node and its *delay* to consider a node as invalid and asks it to go to the inactive mode in order to reduce bandwidth consumption.

Figure 2.14 depicts the overall classes, properties, and relations for our Node Data Model.

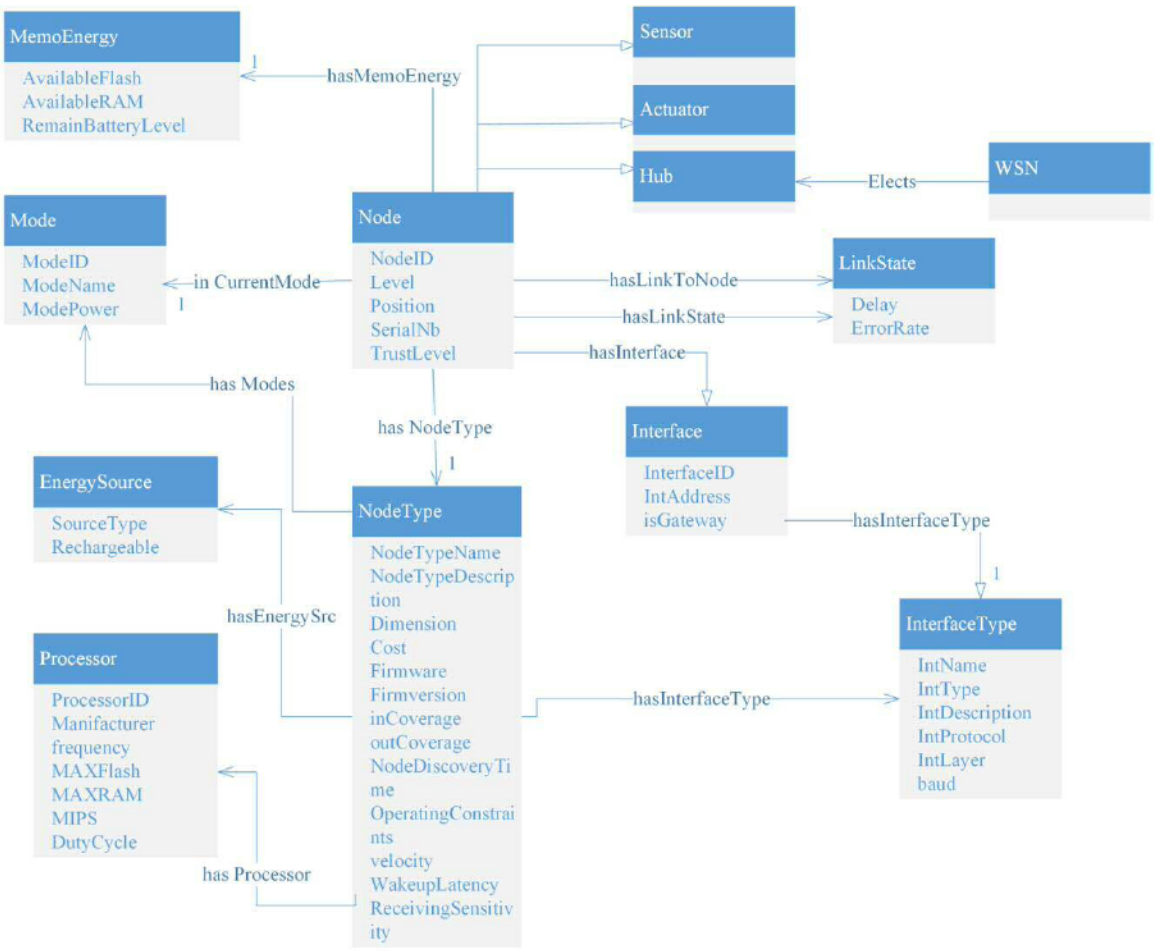

Figure 2.14- Node Data Model.

2.3.3 Process & Measurement Data Model

Each node is used for certain process (temperature, blood pressure, insulin regulation, etc.). The process is already described in almost all existing data model. While sensors measure some measurements, actuators do certain actions. Based on the survey done in section 2.2, we added the following properties to the process: calibration, precision, response order, drift, unit of measurement, latency, and resolution and data rate. These parameters enhance the cooperation between different WSNs measuring the same process. For example, if we are monitoring the temperature, regardless the sensor's physical characteristics, the data rate should be around 120 bps with a precision of 8 bits. In addition, these parameters can help in the management of the access to the medium. If a process is characterized by a small latency, it should be given a higher priority to access the medium and send the data to the Hub. Moreover, the process can have one or many data. For example, the tracking process can include latitude, longitude and speed data.

In particular, for WBANs and health care systems, many recent researches have been conducted to describe the electronic health data. As we can see in the survey section 2.2.2, some models are dedicated for services discovery like GATT profiles (45), others focused on the interoperability between different medical and clinical centres. In this Chapter, we are concentrating on the data sent within the WBAN and sent form the Hub to an external medical control and monitoring server.

EDAM (EMBRACE Data and Methods) (47) is a data model of bioinformatics operations (tool, application, or workflow functions), types of data, topics (application domains), and data formats. The applications of EDAM are within organizing tools and data, finding suitable tools in catalogues, and integrating them into complex applications or workflows. EDAM includes 4 main sub-ontologies of concepts:

- ***Operation***: A function or process performed by a tool; what is done, but not (typically) how or in what context.
- ***Data***: A type of data in common use in bioinformatics,
- ***Topic***: A general bioinformatics subject or category, such as a field of study data, processing, analysis or technology,
- ***Format***: A specific layout for encoding a specific type of data in a computer file or message.

EDAM can be used in our proposed model in case where the biomedical sensors are used for bioinformatics data, for example in the analysis of mutations in cancer. In such application, the process will have as data the data class in EDAM. WSNs are widely used for remote patient monitoring and non-medical applications. That's why we searched for other existing models that describe the data types sensed by biomedical nodes. As shown in the survey section 2.2.2 and in the reference model of the electronic healthcare, the data types include all the biomedical data formats. Whereas the process has one or many data, the data has one data format or type as described in the data value of the reference model. Moreover, we added two properties to the data class: QoS, Data size and Data transmission. The data size reflects the precision and the resolution of the sensed data. In addition, WSNs are characterized by heterogeneous data that differ in their data rate, bandwidth, data types, accuracy, delay and others. An effective QoS mechanism is needed in order to deliver adequately this various information. Besides QoS, data transmission can be unicast, multicast or broadcast. In general, it is considered as multicast when it is sent to group of nodes within a cluster, and as broadcast when it is sent to the entire active nodes in the WSN. If the remaining lifetime of a node is crucial, it can ignore a broadcast or multicast data to save power. The data format can be described by external data model dedicated for each domain of application.

In addition, data have constraints. In SensorML (3), data can be constrained by *securityConstraints* based on the Security Banner Marking model of the Intelligence Community Information Security Marking (IC ISM) Standard, *validTime* that describes the validity time for certain process and *legalConstraints* which can be a private ACT, intellectual Property, rights or copyrights. Otherwise, in the GATT service profiles (45), each data is described by its minimum value and maximum value. Furthermore, data can have operational constraints. Consequently, we proposed that the constraints could be legal constraint, validity constraints (having minimum

and maximum values), operational constraints, security constraints (security protocol and used key or certificate's reference), and compression constraint (compression algorithm).

Finally, we added a light class to store the measurements done by the sensors. One of the main goals of this class is to be implemented within the sensor without affecting its performance permitting data aggregation and data value verification within the node before sending the data to the Hub. It includes the sensed value, the timestamp and can include TTL (Time to live) in order to assure the freshness of the data. In addition, *valid* Boolean property is added to indicate the validity of a measurement. For example, in the following cases, the valid property will be false:

- If the TTL is zero, which means that the measurement is not fresh,
- When the value of a measurement exceeds the maximum range,
- When the measurement is captured before the wakeup latency.

In that way, if the connection to the hub is temporary unavailable, the sensor can check for the freshness and the validity of the sensed data, and if it is expired or is invalid, the data will not be sent to the Hub. Therefore, the valid data property is introduced in our model and used to determine if the measurement is valid or invalid.

Due to the limited resources of the sensors in terms of processor, memory and battery capacity, only this part of the model could be treated by the sensors allowing the conversion from raw data to aggregated data when it is necessary. For example, if only the average heart rate of a patient is needed to calculate the number of burned calories, this calculation can be done by the sensor and sent to the hub without degrading the performance of the node. Fig. 2.15 describes the classes, properties and relations for the Process & Measurements data model.

After the conceptualization of the detailed UML class diagrams, we choose to formalize our model in ontology form (.owl) called MyOntoSens ontology. We insisted on adding the cardinality restrictions, the object properties description (inverse & functional property), and the data properties description (functional). Additional rules related to WSNs are created to infer additional knowledge from basic knowledge. The following sections detail MyOntoSens ontology.

2.4 MyOntoSens Ontology

Benefiting from the properties in OWL DL, we created our ontology as follow. Firstly, we proposed to split and represent the WSN domain with three different sub-ontologies: WSN, Nodes and Process. Those sub-ontologies are fully mirroring the three main classes of our MyOntoSens open data model concept (see section 2.3). The classes' distribution and description are also mirroring the one that were already described in our UML class diagrams presented in section 2.3. Furthermore, when defining the properties of each class, we noted that many properties are variables described by their value and unit of measurement (like e.g. frequency, weight, height, speed, etc.). This pushes us to introduce a new class Quantity and search for an existing class that is capable of generically defining any kinds of units. The choice was to use the Unit class from QUDT (Quantities, Units, Dimensions and Data Types) generic model (48). QUDT was developed by NASA and its last specification and version were released is in April 2013. The proposed model for managing hardware heterogeneity in wireless sensor network (see

survey section 2.2.1) also proposed to use the QUDT ontology for units definition due to its wide adoption in many domains like e-commerce platform. Figure 2.16 depicts a part of the hierarchy of the unit class. This hierarchy is also extensible for new units' type definitions.

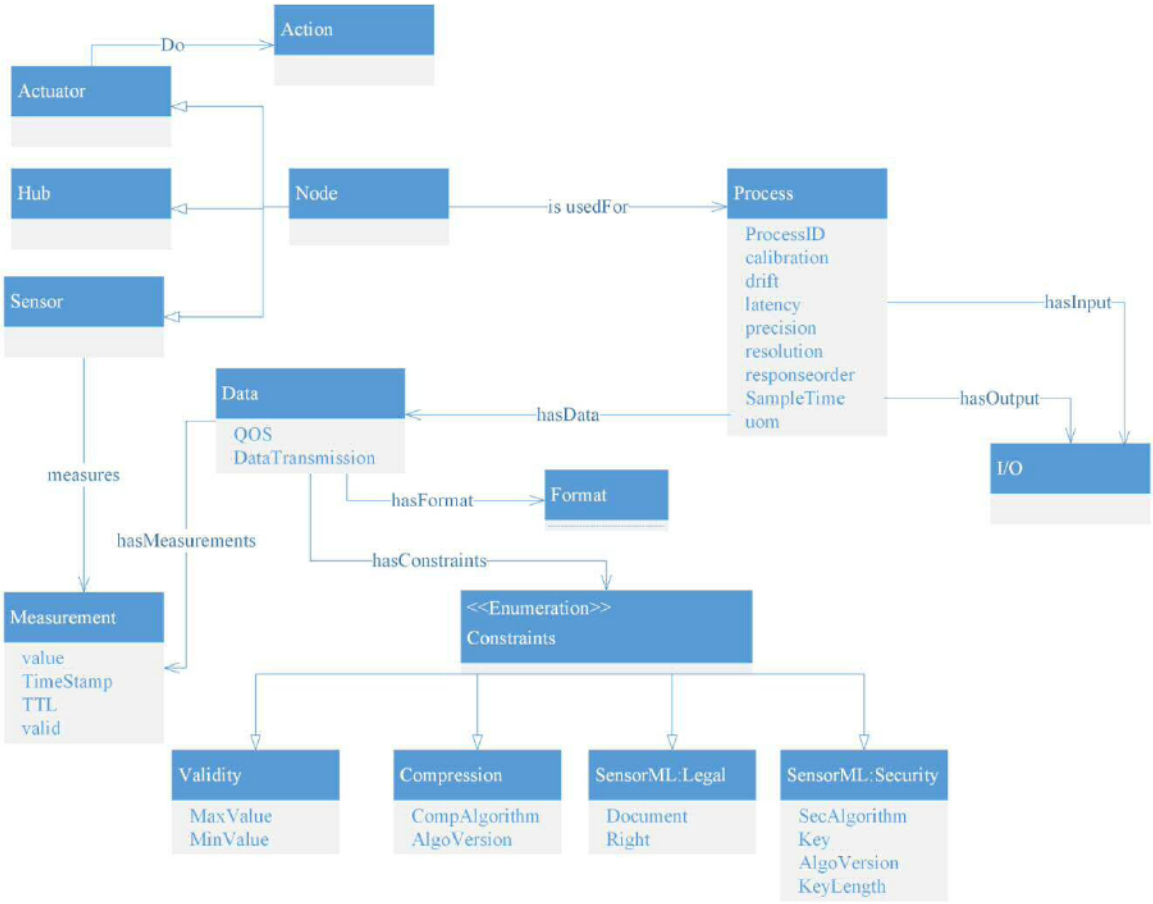

Figure 2.15-Process & Measurements Data Model

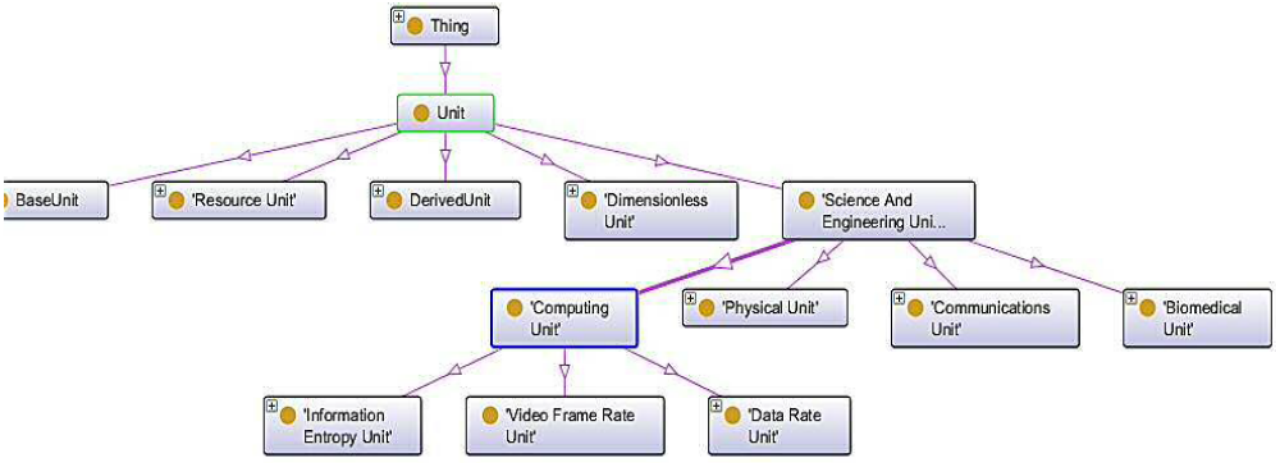

Figure 2.16-Part of the hierarchy of the QUDT unit class

So, each Quantity has as data property *hasvalue* of any type and has, as quantity, an individual of type Quantity.

The node location is essential in WSN where the sensor's position could be: implanted or inside body, body surface or external. In addition, we also have the WSN's location. That is why, we introduced a new class "Location" that has as subclasses:

- Implant: which can be under the skin or inside the human body,
- Body surface: which has as data properties the position of the node relatively to the human body,
- External: this subclass can be used to express the location of the WSN's Hub, or nodes built in the PDA or Mobile of the patient, or to describe the location of the WSN. This location can be given as coordinates for example using the latitude and longitude, or given by an address (City, country, building…).

The use of Location as an independent class gives the user the ability to extend it by importing existing ontologies dedicated for locations. Some of these ontologies are e.g.:

- Location.owl[6], an ontology to describe geographical and non-geographical locations,
- The Places ontology[7], an ontology created by Smethurst, Styles and Scott for describing places of geographic interest.

Figure 2.17 shows the hierarchy of the proposed Location class.

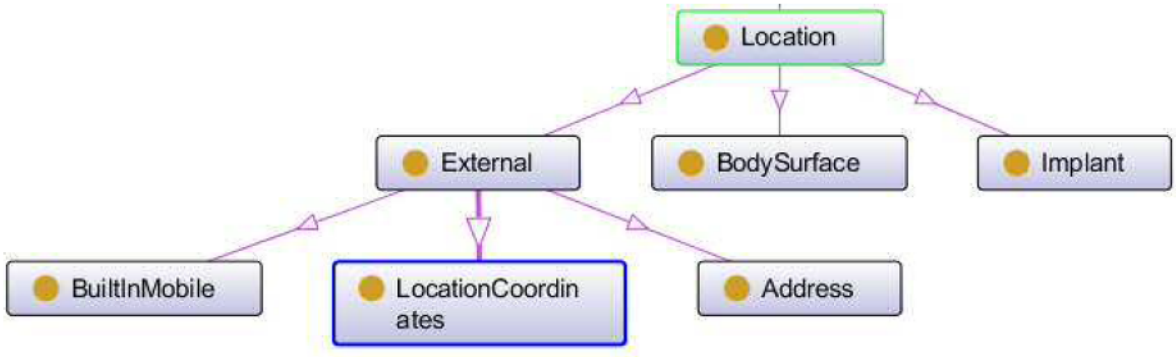

Figure 2.17-Classes hierarchy for Location class

The following paragraphs will now detail our proposed ontology, with all the classes introduced, their objects' properties, and their data properties.

2.4.1 MyOntoSensWSN Ontology

Classes:

Figure 2.18 depicts the general class hierarchy of MyOntoSensWSN ontology.

[6] http://dpedia.org/ontology/location

[7] http://vocab.org/places/schema.html

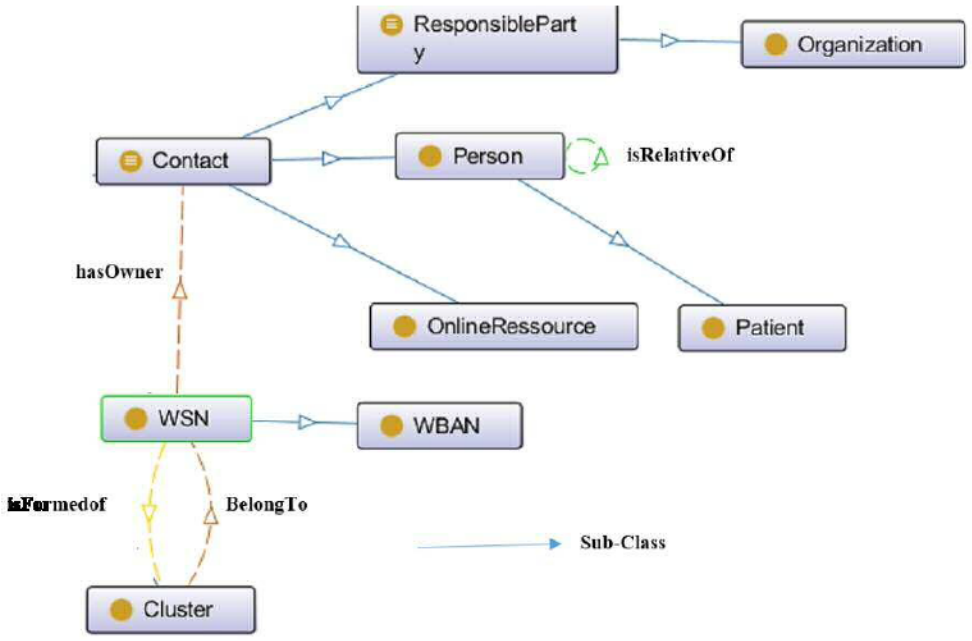

Figure 2.18-MyOntoSensWSN Ontology classes

Object Properties:

Table 2.4-MyOntoSensWSN Object Properties

Object property	Domain	Range	Cardinality		Additional properties
			Max	min	
hasContact	WSN	Contact			
hasOwner	WSN	Patient			*Sub-property* of hasContact
belongTo	Cluster	WSN	1	1	*Inverse of* isFormedof
isFormedof	WSN	Cluster			*Inverse of* belongTo
ElectHub	Cluster	MyOntoSensNode:Hub			
hasCommunicationProcess	WSN	CommunicationProcess			
CommunicationPeriod	Periodic	MyOntoSens:Quantity			
Height	Patient	MyOntoSens:Quantity			
Weight	Patient	MyOntoSens:Quantity			
HipCirumference	Patient	MyOntoSens:Quantity			
RestingHR	Patient	MyOntoSens:Quantity			
MaxHR	Patient	MyOntoSens:Quantity			
WaistCircumference	Patient	MyOntoSens:Quantity			

Data Properties:

Table 2.5-MyOntoSensWSN Data Properties

Data Property	Domain	Range	Additional property
Age	Patient	Int	<100
ApplicationDomain	WSN	String	
Density	WSN	String	
DeploymentTime	WSN	dateTime	
Email	Person	String	
MaxFatBurnHR	Patient		
MinFatBurnHR	Patient		
FaultTolerence	WSN	Real	
Gender	Person	String	length 1
Language	Person	String	
Link	OnlineRessource	anyURI	
Phenomena	WSN	String	
Phone	Person or		
RadioTech	WSN	String	
Topology	WSN	{"Adhoc","Stars","Mesh","P2P","Others"}	
UserID	Person		
WSNID	WSN	anyURI	Functional
ClusterID	Cluster	anyURI	Functional
Channel	Cluster		
NbNodes	Cluster	Int	

2.4.2 MyOntoSensNodes Ontology

Classes:

While Figure 2.19 depicts the hierarchical structure of the node class, Figure 2.10 depicts the relations between a node and other classes.

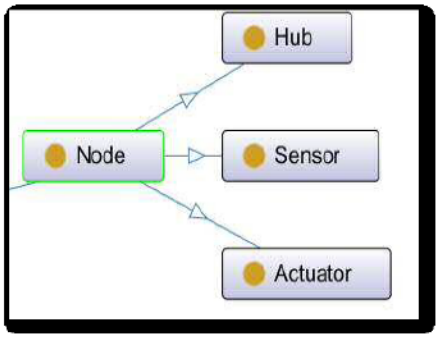

Figure 2.19-Node's subclasses

Object Properties:

Table 2.6-MyOntoSensNodes Object Properties

Object property	Domain	Range	Cardinality	
			min	max
Cost	NodeType	MyOntoSens:Quantity		1
dimension	NodeType	MyOntoSens:Quantity		4
DutyCycle	Processor	MyOntoSens:Quantity		1
frequency	Processor	MyOntoSens:Quantity		1
hasAvailableFlash	MemoEnergy	MyOntoSens:Quantity		1
hasAvailableRAM	MemoEnergy	MyOntoSens:Quantity		1
hasEnergySrc	NodeType	EnergySource		
hasInterface	Node	Interface		
hasInterfaceType	Interface	InterfaceType	1	1
hasLinkState	Node	LinkState		
hasLinkToNode	LinkState	Node		1
hasMAXFlash	Processor	MyOntoSens:Quantity		1
hasMAXRAM	Processor	MyOntoSens:Quantity		1
hasMemoEnergy	Node	MemoEnergy		1
hasModes	NodeType	Modes		
hasNodeType	Node	NodeType		1
hasProcessor	NodeType	Processor		
hasRemainBattery	MemoEnergy	MyOntoSens:Quantity		1
inCoverage	NodeType	MyOntoSens:Quantity		
inCurrentMode	Node	Modes	1	1
MIPS	Processor	MyOntoSens:Quantity		1
ModePower	Mode	MyOntoSens:Quantity		
NodeDiscoveryTime	NodeType	MyOntoSens:Quantity		1
outCoverage	NodeType	MyOntoSens:Quantity		
Position	Node	MyOntoSens:Location		
receivingSensitivity	NodeType	MyOntoSens:Quantity		1
velocity	NodeType	MyOntoSens:Quantity		1
WakeupLatency	NodeType	MyOntoSens:Quantity		1

Data Properties;

Table 2.7-MyOntoSensNodes Data Properties

Data Property	Domain	Range	Additional property
InterfaceAdress	Interface		
Int_Type		string	SubProperty of InterfaceAddress
Int_value			SubProperty of InterfaceAddress
baud	InterfaceType		
delay	LinkState	float	
ErrorRate	LinkState	float	
Firmversion	NodeType	string	
Firmware	NodeType	string	
IntDescription	InterfaceType	string	
InterfaceID	Interface	anyURI	Functional
IntLayer	InterfaceType	string	
IntName	InterfaceType	string	
IntProtocol	InterfaceType	string	
isGateway	Interface	boolean	
ModeID	Mode	anyURI	Functional
ModeName	Mode	string	
NodeID	Node	anyURI	Functional
NodeTypeDesciption	NodeType	string	
OperatingConstraints	NodeType	string	
ProcessorID	Processor	anyURI	Functinal
Rechargeable	EnergySource	boolean	
SerialNb	Node	string	
SourceType	EnergySource	string	
TrustLevel	Node		

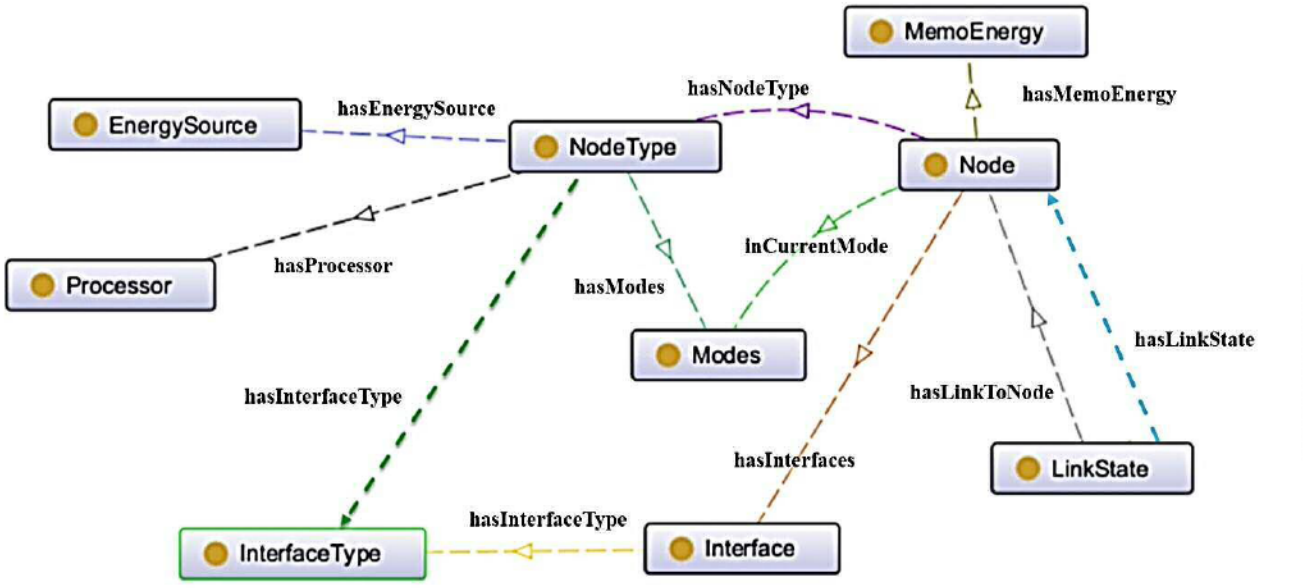

Figure 2.20-Nodes Classes Relationships

2.4.3 MyOntoSensProcess Ontology

Classes:

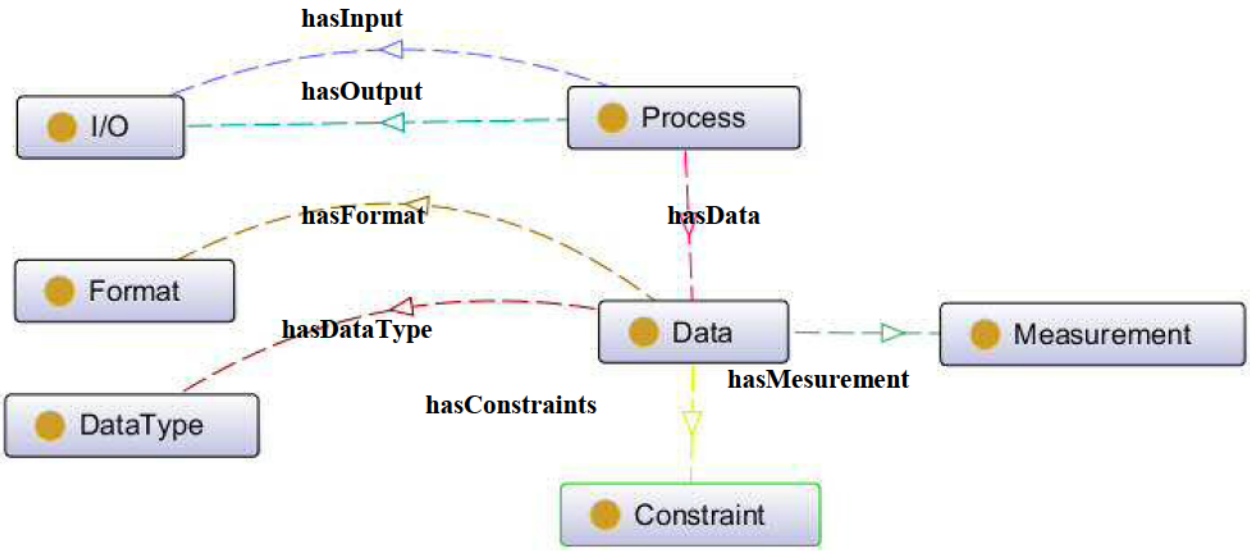

Figure 2.21-Process class hierarchy

Object Properties:

Table 2.8-MyOntoSensProcess Object Properties

Object property	Domain	Range	Cardinality	
			Min	max
calibration	Process	MyOntoSens:Quantity		1
DataSize	Data	MyOntoSens:Quantity		1
DoAction		Action		
Drift	Process	MyOntoSens:Quantity		
hasConstraints	Data	Constraints		
hasData	Process	Data		
hasDataType	Data	DataType		1
hasDerivedData	Process	Data		
hasFormat	Data	Format		1
hasInput	Process	I/O		
hasMesurement	Data	Measurement		
hasOutput	Process	I/O		
latency	Process	MyOntoSens:Quantity		1
precision	Process	MyOntoSens:Quantity		1
resolution	Process	MyOntoSens:Quantity		1
responseorder	Process	MyOntoSens:Quantity		1
SampleRate	Process	MyOntoSens:Quantity		1
Uom	Process	Unit		1

Data Properties:

Table 2.9-MyOntoSensProcess Data Properties

Data Property	Domain	Range	Additional property
AlgorithmVersion	Compression or Security	string	
CompAlgorithm	Compression	string	
Key	Security	string	
KeyLength	Security	int	
MaxValue	Validity		
MinValue	Validity		
ProcessID	Process	anyURI	Functional
QoS	Data		
SecAlgorithm	Security	string	
TimeStamp	Measurement	dateTime	
TTL	Measurement	int	
valid	Measurement	boolean	
value	Measurement		

2.4.4 MyOntoSens Ontology

MyOntoSens ontology includes four direct imported ontologies: MyOntoSensWSN.owl, MyOntoSensNodes.owl, MyOntoSensProcess.owl and dimension.owl (which contains the unit classes as described by QUDT; this ontology will lead to an indirect import for the qudt.owl ontology).

Classes:

We added the Quantity class, the Unit class and subclasses to our WSN ontology, as already depicted in 2.16 and Location class and subclasses as already presented in 2.17.

Object Properties:

Table 2.10-MyOntoSens Object Properties

Object property	Domain	Range	Cardinality		Additional property
			min	max	
BelongTo	Node	WSN		1	*Inverse of* Contains
Contains	WBAN	Node			*Inverse of* BelongTo
DoAction	Actuator	Action			
ElectHub	WSN	Hub		1	
hasUnit	Quantity	Unit			
isMeasuredBy	Measurement	Sensor		1	*Inverse of* measures
measures	Sensor	Measurement	1		*Inverse of* isMeasuredBy
usedFor	Node	Process	1		
BelongToCluster	Node	Cluster		1	*Inverse of* ComposedOf
ComposedOf	Cluster	Node			*Inverse of* BelongToCluster

Data Properties:

Table 2.11-MyOntoSens Data Properties

Data Property	Domain	Range	Additional property
hasValue	Quantity		

Rules:

Additional rules are added to the ontology based on the domain of WSN's applications. These rules are defined using Semantic Web Rule language SWRL.

Rule 2.1: When a node measures some measurements having certain Data belonging to a process A, we should assume that this node is used for process A. the rule is:

Node(?N), measures(?N, ?M), hasData(?P, ?D), hasMesurement(?D, ?M) -> usedFor(?N, ?P)

Rule 2.2: When a WSN elects a Hub, we have to infer that this WSN contains the Hub.

WSN(?W), ElectHub(?W, ?S) -> Contains(?W, ?S)

Rule 2.3: If the WBAN contains a sensor built in the mobile, and the WSN elects this mobile as Hub, we should assert that the sensor has the same node type as the mobile.

WSN(?W), Position(?N, BuiltInMobile), hasNodeType(?S, ?NT), Contains(?W, ?N), ElectHub(?W, ?S) -> hasNodeType(?N, ?NT)

Rule 2.4: If the TTL of a measurement is equal to zero, the measurement should be invalid.

Measurement(?M), TTL(?M, ?T), equal(?T, 0) -> valid(?M, false)

Rule 2.5: In the context of WBAN, the hub has to observe the link state of the nodes belonging to the same WBAN of a certain patient. Thus, if a Hub has link state to a node, and this Hub belongs to a WBAN, the node should belong to the same WBAN.

WBAN(?W), hasLinkState(?S, ?L), hasLinkToNode(?L, ?N), Contains(?W, ?S) -> Contains(?W, ?N)

WBAN(?W), hasLinkState(?S, ?L), hasLinkToNode(?L, ?N), Contains(?W, ?N) -> Contains(?W, ?S)

Rule 2.6: if the owner of a WBAN has relatives, these relatives should be considered as reference contacts to the WSN.

WBAN(?W), hasOwner(?W, ?O), isRelativeof(?O, ?R) -> hasContact(?W, ?R)

Rule 2.7: These rules determine the family relationships between the patient and his relatives.

Person(?P), Sister(?P, ?P1) -> Sister(?P1, ?P)

Person(?P), Children(?P, ?P1) -> Parent(?P1, ?P)

Person(?P), Friend(?P, ?P1) -> Friend(?P1, ?P)

The users of this ontology can define additional rules related to its application using the SWRL. We have just defined the general rules applicable for all types of WSN/WBAN applications.

2.5 MyOntoSens Pre-validation : OntoRun Mobile Application

The objective of this section is to present the first pre-validation tests of the MyOntoSens ontology. Mainly based on symbolic validation methods, those tests will demonstrate the a priori validity of the proposed MyOntoSens ontology.

While exercising is becoming primordial in our daily life, various mobile applications were developed for runners capturing vital signals from sensors in order to calculate the number of burned calories. However, these applications still lack for compatibility and interoperability with other systems. Thus, to pre-validate our ontology, an Android mobile application for calories burned during running exercise was implemented. This mobile application is fully relying on the

MyOntoSens ontology that is introduced for overcoming the compatibility and interoperability issues aforementioned.

Figure 2.22 depicts the overall architecture of the application. It is a flat architecture where the WBAN elects the user's PDA as Sink. The runner is equipped with the H7 Polar sensor[8] for heart beat measurements. The smart phone collects the heart beats from the sensor via Bluetooth LE communication, and creates individuals related to MyOntoSensProcess ontology. Moreover, the smart phone will send the necessary data to the semantic cloud server where the full MyOntoSens ontology resides. The user receives notifications in case he/she exceeds the maximum heart rate 'normal range. The detail of the semantic individuals, as well as the process of OntoRun application, are given in the next sub-sections.

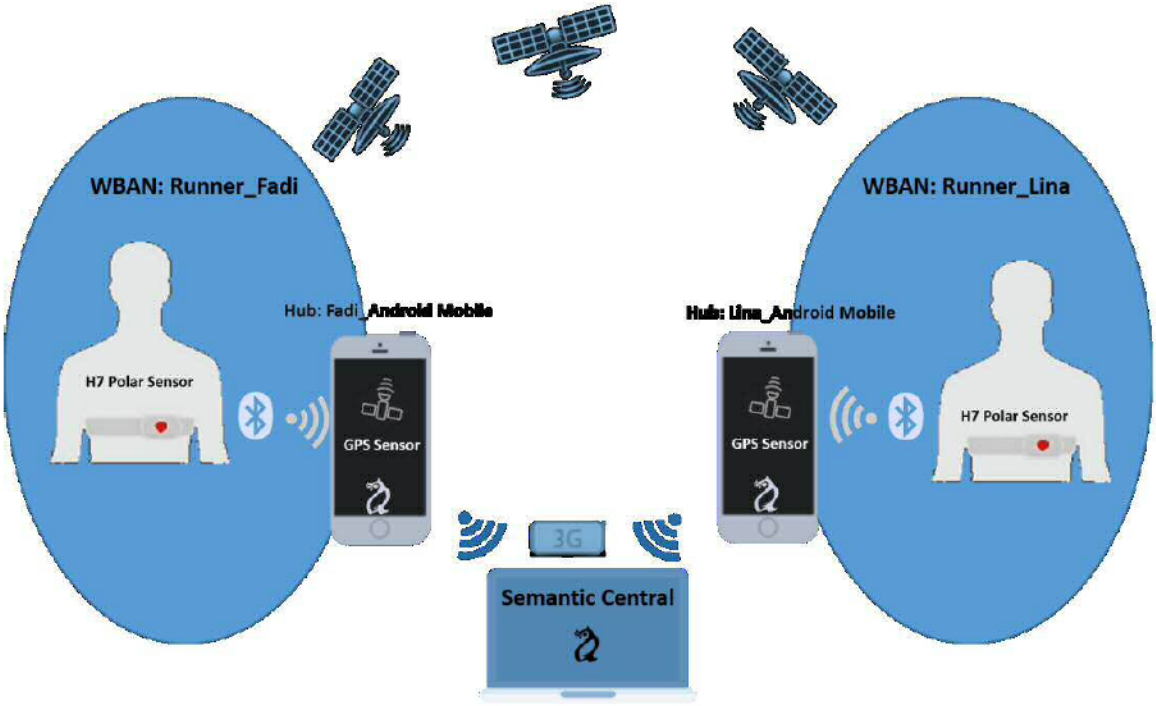

Figure 2.22-High level architecture of OntoRun application

2.5.1 OntoRun Individuals in Protégé

To pre-validate the ontology, we have taken the example of a WBAN equipped with a heartbeat sensor and GPS receiver in order to calculate the calories burned of a runner during exercises. For testing purposes, we created two WBANs *Runner_Fadi* and *Runner_Lina*, as depicted in Figure 2.22. Because the WBANID is a functional property, each WBAN can have only one WBANID that's why we have chosen the MAC address of the mobile where the application is downloaded knowing that the application can be downloaded once on the mobile. The WBAN has the runner, identified by his email address, as contact. In order to calculate the burned calories, the runner should be described by his age, gender, height and weight. The WBAN

[8] http://www.polar.com/en/products/accessories/H7_heart_rate_sensor

contains the GPS and H7 Polar sensors and elects the Mobile as Hub. Figure 2.23 depicts a part of *Runner_Lina's* WBAN ontology instance. The object property, Contains *Lina_AndroidMob* is inferred based on Rule 2.2 defined in section 2.4.4.

The mobile has, as node type, Samsung S4 and is equipped with a Bluetooth LE interface and a 3G interface which connects the gateway of the WBAN to the internet. The GPS sensor is built in the mobile and used for tracking process. The H7 Polar sensor communicates with the mobile via the Bluetooth LE interface. The H7 Polar is identified by the Serial number retrieved from its GATT general attributes (see survey section 2.2.2) and is used for the heart beat process. The H7 Polar has the waist as position (individual of type *BodySurface*, the subclass of the location class).

While the tracking process has the meter (m) as unit of measurement, the heart beat process has beeps per minutes (bpm) as unit. The tracking process has two data, the latitude and longitude, and an additional one derived from the speed calculated based on Haversine formula. The heart beat process has the HR data. Each data has measurements measured by a sensor.

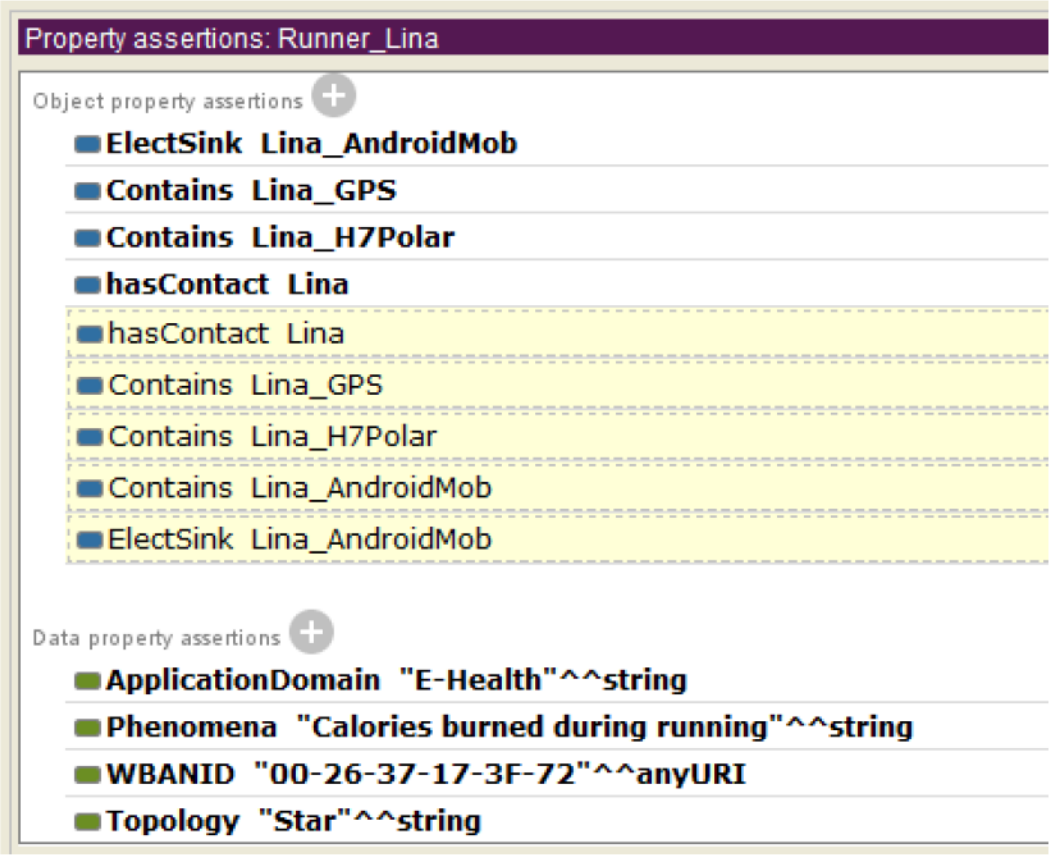

Figure 2.23-Part of Runner_Lina's WBAN ontology instance

To encourage user to run on higher speed, we are calculating the maximum heart rate based on the following formula: Maximum (51)

Thus, we added the following SWRL Rule 8.

Rule 2.8: Patient(?P), MaxHR(?P, ?Q), Age(?P, ?Age), multiply(?x, 0.7, ?Age), subtract(?y, 208, ?x) -> hasValue(?Q, ?y).

If the H7 Polar measures a value greater than the maximum heart rate, a notification will be sent to the user and the measurement will be considered as invalid in order to stop the running process. To do so, the SWRL rule 2.9 has been added.

Rule 2.9: WBAN(?W), MaxHR(?P, ?Q), hasContact(?W, ?P), Contains(?W, ?S), measures(?S, ?Mes), usedFor(?S, HeartBeat), value(?Mes, ?mesure), hasValue(?Q, ?max), lessThan(?max, ?mesure) -> valid(?Mes, false).

Figure 2.24 shows the inferred object properties of Lina_GPS sensor:

- *BelongTo Runner_Lina*: inferred from the inverse property Contains (*Lina_WBAN* Contains *Lina_GPS*).
- Measures: inferred from the inverse property is*MeasuredBy* (e.g *Lina_Long2* is *MeasuredBy Lina_GPS*)
- *usedFor*: inferred from Rule 2.1 already defined in section 2.4.4.

The maximum heart rate for Lina should be 187. Thus, if H7 Polar measures a value greater than this maximum, the measurement should be invalid. Figure 2.25 shows an invalid measurement.

Using Pellet reasoner, we verified that the ontology is consistent and well classified. The inferred value was retrieved. More rules and verification can be done to assure the well functionality of the proposed ontology. To retrieve the information, we have used the SPARQL engine implemented in Protégé. Unfortunately, the SPARQL engine in Protégé cannot retrieve inferred data. Here is the example of the written query for node discovery:

```
SELECT ?WBAN ?Nodes ?Hub
        WHERE { ?WBAN full:Contains ?Nodes.
?WBAN full:ElectHub?Hub.
}
```

The result of the query is given in Figure 2.26.

It is now the time to go to the real implementation of OntoRun mobile application.

2.5.2 OntoRun Mobile Application

To test our ontology in a real environment, we developed an Android application that receives the heartbeat of the runner from H7 Polar sensor via Bluetooth LE interface. Fig. 2.27 depicts the general architecture of our m-health application. Unlike the pre-validation scenario, we divided the WBAN into cluster allowing extending the flat architecture to a multi-tiers architecture where more than one cluster can be used and one or more hubs can be elected. For example, this scenario can be used not only for runners outside their home, but also for runners using running machine within their home equipped with temperature sensor, humidity sensor, etc. In this case, the WSN can encompass another cluster which is the home cluster that can elect as sink a local

server. Moreover, the H7Polar cluster can be formed of more Bluetooth LE sensors like pedometers and accelerometers to calculate accurately the runner's speed.

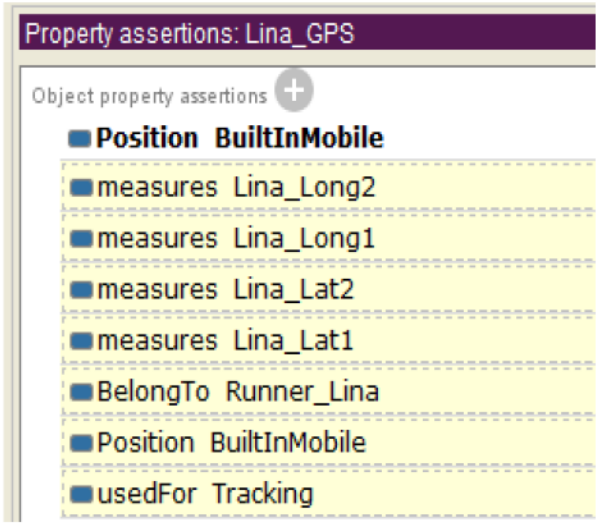

Figure 2.24-GPS Object properties

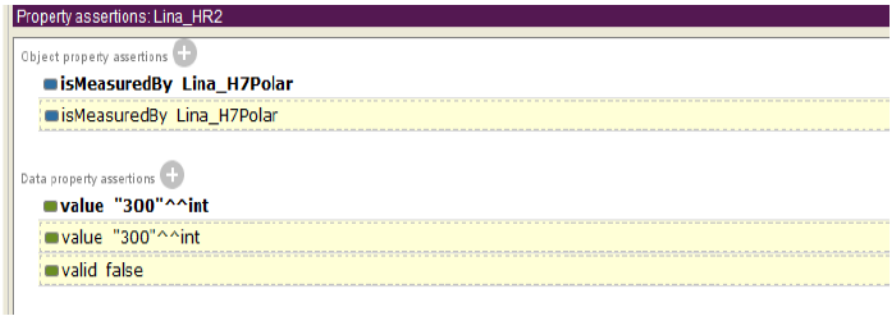

Figure 2.25-Invalid measurement caused by a value greater than the maximum limit

WBAN	Nodes	Hub
Runner_Fadi	Fadi_GPS	Fadi_AndroidMob
Runner_Fadi	Fadi_H7Polar	Fadi_AndroidMob
Runner_Lina	Lina_GPS	Lina_AndroidMob
Runner_Lina	Lina_H7Polar	Lina_AndroidMob

Figure 2.26-Node discovery SPARQL result

For our implementation, the Jena API (23) is used on the server side to create individuals and infer the entire model. The AndroJena [9], dedicated for Android mobile phone, helped us to create individuals on the mobile side and retrieve the inferred schema.

[9] http://code.google.com/p/androjena/

Once the runner downloads the application, he/she is authenticated using the Google plus API (this was only retained for implementation simplicity purposes; security is out of the scope of our testing case). The mobile then sends its MAC address (used to identify the runner's WSN), the user's Gmail account (used to identify the Contact), the user's age, the user's gender and the user's weight (used as data properties of the Contact class) to the semantic server (e.g. remote control and monitoring centre) via 3G interface. The user related data are transferred via JSON format (32) because of its lightness in processing and its non-intensive bandwidth utilization. Therefore, the Server creates a new WSN:

- Having as contact the user.
- And made of the Mobile Cluster comprising the GPS sensor built in the mobile. When the user turns on the H7 Polar, the mobile detects its presence and informs the server that a new cluster has been created and has elected the mobile as Sink.

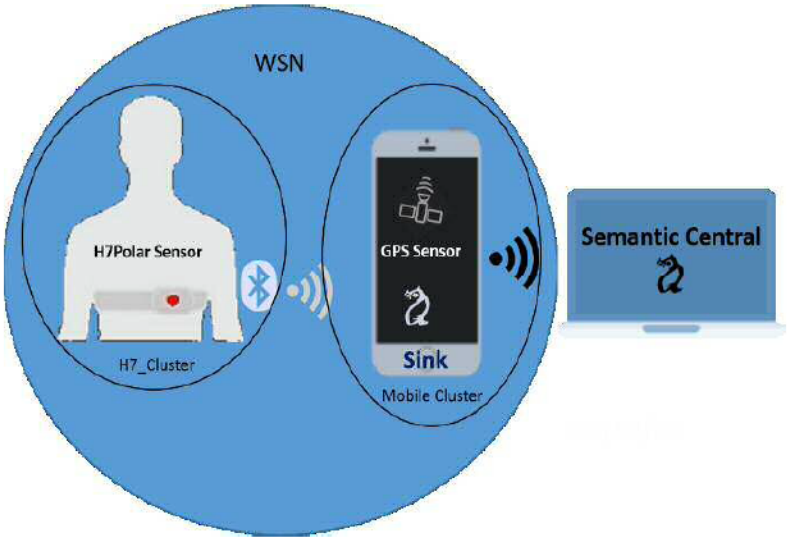

Figure 2.27-General Architecture of the testing environment

The user can now start a new task. During exercises, the HR data, the burned calories and the attempted distance are displayed in real time. The burned calories are calculated based on the following equations (49):

Where HR = Heart rate (in beats/minutes)
W = Weight (in kilograms)
A = Age (in years)
T = Exercise duration time (in hours)

The user can also discover the nearby runners and check the history of his activities. For the discovery of neighbouring runners, the following SWRL query (Query 1) has been carried out:

SELECT ?Contact

WHERE { ?WSN w:hasContact f:Runner1. ?WSN f:hasLatitude ?lat. ?WSN f:hasLongtitude ?long. ?lat rdf:type w:Quantity; p:hasvalue ?Runner1_lat. ?long rdf:type w:Quantity;p:hasvalue ?Runner1_long. ?WSN1 w:hasContact ?Contact. ?WSN1 w:isFormedOf ?Cluster. ?Cluster n:composedOf ?N. ?N n:inCurrentMode f:Active.?WSN1 f:hasLatitude ?lat1.

?lat1 rdf:type w:Quantity; p:hasvalue ?val1. ?WSN1 f:hasLongtitude ?long1. ?long1 rdf:type w:Quantity; p:hasvalue ?val2.

Filter (((?Runner1_lat - ?val1) (?Runner1_lat - ?val1) + (?Runner1_long - ?val2) * (?Runner1_long - ?val2)) >0.05)*

} (Query 1)

If the HR exceeds the maximum range, the user will receive a notification in order to reduce his speed. Figure 2.28 depicts the general process of OntoRun mobile application.

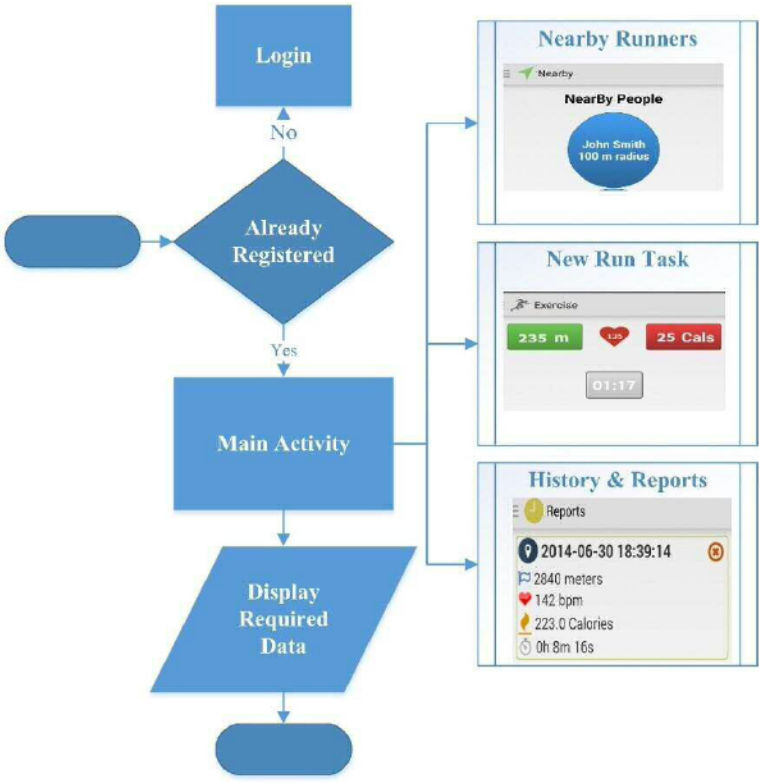

Figure 2.28-General architecture of the m-health application

We tested our application on many users equipped with an H7 Polar sensor and having Android mobiles supporting Kitkat version in order to be able to use the embedded Bluetooth LE interface. The automatic node discovery on the server side was successfully accomplished. The neighbours were detected due to the use of SPARQL query.

2.5.3 Benefits of MyOntoSens Ontology

The proposed m-health application demonstrates the compatibility of our ontology with Bluetooth LE GATT profiles that were just directly mapped into this ontology. Our implemented application is able to accept any other Bluetooth LE enabled sensor, from any vendor, without any reprogramming (thanks to our MyOntoSens ontology). Furthermore, the *Format* class that is introduced in our model can be linked to any external data format. In that way, vendors and developers carry out their own applications and services based on our ontology will make them interoperable with any data types.

Due to the use of:

- Extended semantic attributes of MyOntoSens ontology (in particular *CurrentMode* and *Location*),
- Additional object properties (in particular *isFormedOf* and *hasContact*) and rules added within our data model,
- And SPARQL rules and queries,

we were able to provide automatic sensor discovery, as well as automatic neighbouring WBANs discovery within both our remote monitoring /control server (semantic central depicted in Fig.2.27) and our mobile application.

Integrating semantic sensor data into the web, due to the use of IRI[10], alleviated the reusability of data by different applications. In our proposed model, the *WSNID*, the *ClusterID*, the *NodeID*, *the ProcessID* and the *InterfaceID* are defined as "anyURI", which means that we are also giving an IRI to all of them for providing a shared access to the associated information through the web (via e.g. SPARQL queries). Taking the example of biomedical data, it is essential for different medical applications and servers to share data in order to detect severe diseases or anomalies. Moreover, it is useful for a patient if services and applications share the same data, thus eliminating the need to fill redundant information about the patient.

Our tests also pointed out that being able to define a generic ontology covering the most important features of WSNs not only provides a solution for interoperability and heterogeneity management, but allows the reduction of the power consumption of the WSN. Indeed, we are only sending few data to the remote monitoring and control server due to the carrying out of local data processing operations directly within the sinks. This was feasible thanks to both the extended semantic knowledge introduced by our ontology within the sinks (i.e. the mobile phones in our use case) and the use of simplified SWRL rules and inference engines on that local semantic knowledge. These simplified inference operations were performed without any noticeable extra energy consumption. As an illustration, once the H7 Polar is on, the mobile only sends that *H7 Polar is in active mode*, with its MAC address. On the server side, the creation of H7 Polar as cluster electing the user's mobile as sink automatically infer the fact that this sensor belongs to this user (runner). Moreover, by creating the object property describing that the sensor has H7 Polar as *NodeType*, we enable the retrieval of all the node's characteristics (processor,

[10] http://www.w3.org/International/O-URL-and-ident.html

interfaces, energy source, etc.). In addition, using SWRL rules and thanks to SPARQL query, the mobile can retrieve the burned calories with minimum processing.

Finally, we are using JSON-LD instead of an XML-based protocol to exchange data between the sink (i.e. the mobile phone in our use case) and the server (i.e. the semantic central in our case), which reduces the bandwidth usage. Indeed, JSON has simpler information modelling and message structures than XML-based mechanisms. Furthermore, JSON is also low processing oriented, which makes it better adapted to devices with limited capabilities such as e.g. cluster sinks or sensors.

2.6 Conclusion

In this Chapter, we have proposed a new semantic open data model and ontology, named MyOntoSens, which generically describes any kind of components of these sensor networks. In addition to the introduction of extended semantic knowledge, compared to existing sensor network ontologies, MyOntoSens provides a solution for WSNs interoperability management, as well as WSNs/sensors data heterogeneity management. It also eases and improves the management process of WSNs, including data collection, aggregation and fusion, as well as routing, control and monitoring. Unlike previous models, MyOntoSens ontology ensures semantic interoperability between nodes and WSNs. Furthermore, to offer scalability and lightness, MyOntoSens was structured in several parts that can be handled and instantiated either in the sensor nodes, the cluster sink(s) or in any other distributed management and control entity (like e.g. within a private and dedicated cloud).

As pre-validation use case, we took the example of a running exercise application that calculates the burned calories during a defined task and alerts the runner if he exceeds certain speed or if his HR exceeds the maximum threshold. In addition, our application can inform the runner about the nearby WBANs and runners. Each runner was equipped with a smart heart-rate sensor, a Smartphone as data aggregator and sink, and a GPS sensor for location tracking and speed calculation (in order to deduce the burned calories).

To ensure syntax checking and consistency checking, Pellet reasoner was used. Within Pellet, all the inferred properties were deduced and our proposed ontology was well classified, which confirms its validity. MyOntoSens was then qualified in a real environment comprising an H7 Polar heart-rate sensor and an Android mobile phone provided with a GPS sensor. Our running exercise and control application was fully implemented based on MyOntoSens and installed on the Android mobile phones of runners. The tests that were performed in this environment show the interoperability of MyOntoSens with Bluetooth LE GATT profiles and also prove that the use of inference engine on MyOntoSens in our m-health application reduces the Smartphone's battery usage.

However, more work need to be conducted to finalize the validation of our ontology and to study its scale up, in particular by investigating more complex test scenarios. Furthermore, extended service ontology is required in order to offer an effective WSN management. This upper ontology will use the properties described in MyOntoSens ontology in order to ease the implementation of end user services. For that reason, we have proposed, specified and formalized upper service ontology, described in Chapter 3, as well as a general architecture that:

- Encompasses these ontologies (i.e. MyOntoSens and our proposed upper service ontology),
- Enables the automatic discover of nodes and servers,
- And enhances the management and data collection/fusion in WSNs.

This architecture is detailed in Chapter 4 and validated within a Smart Home scenario for elderly fall detection monitoring.

CHAPTER 3- OPEN SERVICE MODEL for WSN "MyOntoService" Ontology

3.1 Introduction

MyOntoSens Ontology is considered as the first step to resolve the problem of heterogeneity and interoperability issues, by assigning an explicit and common semantic to every terminology used for WSNs, in particular the information concerning the WSN nodes, corresponding data and process. It encompasses the general information needed for WSN's description, nodes' characteristic (software, hardware, memory, processing, interfacing), observed features parameters and measured values. However, our main aim is to create a generic data exchange system in order to enhance the remote monitoring, management and control of wireless sensor networks. In that context, the WSN can be seen as a provider supplying different services for the end users. The application/service developers, who are not essentially sensor experts, make application/service that uses different WSNs form different domains of application and offer flexible services for different users. To achieve this goal, we decided to adopt the SOA strategy for the WSN. Moreover, we should allow the users to access the WSN's components or data from anywhere using simply the Internet which is available widely. If Web Services are the preferred standards-based way to realize SOA, these solutions will not provide an interoperable Layer without the use of semantic solutions. Combining the idea of SOA, Web Services and semantic description delivered our proposed Upper Service Ontology: "MyOntoService".

3.2 Background

In this section we will introduce the SOA characteristics and focus on how it can be beneficent in interoperability and heterogeneity management for WSNs. In addition, the web services and semantic web services are described showing how they can be used for service and sensor discovery in WSNs.

E. Townsend, defined the SOA as the technique of functionally decomposing the application functions of a large organization into a set of interoperable services that can be accessed through "network APIs" (50) . The service is a software component that accomplishes certain task, and can be reused by other service or accessed by an end user via an interface or API. This approach separates the development of service interface from its implementation. SOA provides an opportunity to tackle interoperability barriers between different service providers by offering loosely coupled applications, and enhanced reuse of existing assets and applications. It decreases the cost of application development because we can create each service independently (51).

SOAs also create the ability to replace suppliers more easily by allowing different services to be provided by different systems. In other words they can reduce vendor lock-in and provide natural transition strategies (51).

Technically, SOA uses a find-bind execute paradigm. Its main components are:

- Service providers: publish services that execute certain functionality, using given input and precondition parameters, and produce output and effects.
- Service consumers: use services published by service providers.
- Service registry: a repository where the published services are described to be discovered and accessed by service consumers.

After understanding the principles of SOA, let's see now how it can be used for WSNs. The new trend in WSN is toward allowing publishing the data over the Internet in order to be aggregated and processed for further decision making. Let's consider the example of a WBAN dedicated for elderly assisting. The data should be procured to medical servers and emergency servers to interact when needed. The medical server can request, for example, to locate the patient. Regardless what this medical server is using as technology, he/she should be able to use the indoor localization service. Moreover, he/she can request any data needed for further statistics and correlations. Thus, the WSN nodes will play the role of service providers. Therefore, a service registry should be placed on a gateway node or local server and the external users (medical servers/hosts, emergency servers, and relatives' hosts) will play the role of service consumers.

Existing solutions are in general closed solutions where each provider offers a set of services and the user can choose between them. Going back to the example of the WBAN for elderly assistance, let's consider that the smart home is equipped with indoor localization sensors. And let's consider that a fire occurs in the building where he/she is living. The fireman should locate this elderly to help him/her. Unfortunately, the fireman could not benefit from the indoor localization service provided in the elderly home because existing solutions are closed and data cannot be accessed by users outside the provider's system. Our approach will provide services to Internet authorized users. In that way, the fireman can use the indoor localization service and save the life of the elderly.

It is clear that the integration of SOA architecture for WSN will open new opportunities for service/application developers where they can see the WSN as a service producer. These services can range from simple sensed data monitoring, to more complicated services requiring data aggregation and processing such as alerts, reminders, routing, security, etc. Different developers can exchange their services eliminating the redundancy in services development.

However, the eternal conflict between flexibility and complexity rises here. Because services can invoke each other, a complete input parameters validation is needed by each service. Moreover, the communication between these services may generate a large number of messages. Thus, this architecture may cause an overhead and affect negatively on the system's performance especially when dealing with real time monitoring services. In addition, the granularity and flexibility of this architecture may disturb the testing and maintenance process of the system. New added services can decrease the system's performance. Apparently, giving the opportunity to share services between different developers and service providers will cause interoperability problems. All these constraints should be taken into consideration when adopting a SOA system. Therefore, the necessity arises to espouse an Upper Service Layer. This layer should include semantic description of WSN's services, irrespective of whatever lower layers and radio technologies are used underneath, in order to tackle the problem of interoperability. Moreover, It should not cause overhead especially in such constrained networks. This Upper Service intend to offer the end

user the chance to request the services provided by the WSN transparently like any other web service, as well as the WSN manager to access the WSN components for remote management.

To design this Upper Service Layer we started by reviewing existing ontologies dedicated for web services. WSN is a part of the IoT and M2M networks. Thus we mainly studied the existing ontology describing IoT and M2M networks. The next section will detail these exiting works and highlights how we reused certain ideas in order to conceptualize our Upper Service Ontology.

3.3 Existing Service Ontologies

3.3.1 OWL-S

OWL-S (6), the semantic mark-up for web services, developed by DARPA DAML program using OWL languages aims to describe semantic web services. It permits the automatic web service: discovery, invocation, composition, interoperation, and execution monitoring. The OWL-S ontology includes mainly three sub-ontologies as shown in Figure 3.1: the service profile ontology that describes what the service does; the process model ontology that describes how the service is used; and the grounding ontology that describes how to interact with the service.

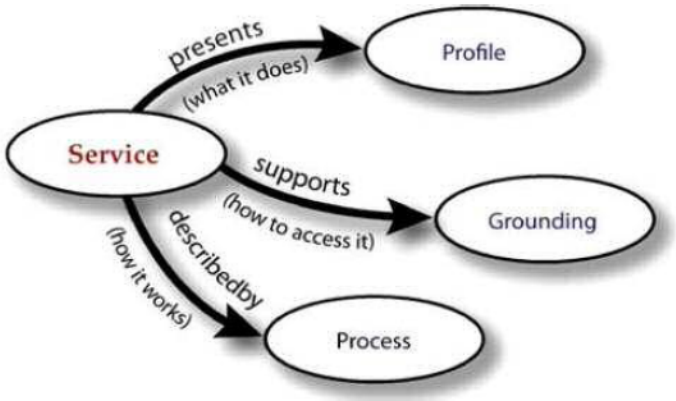

Figure 3.1-Top Level of the Service ontology (6)

3.3.1.1 Service Profile

One of the main aims of the service profile is to provide the automatic service discovery. On one hand, when a producer wants to publish a server, it should describe the service based on the service profile ontology. On the other hand, the requester will discover the capabilities of available servers based on the service Profile. The service profile includes three different aspects:

- Functional aspect that determines the functionality of a service. The "*hasProcess*" object property, connects a "*Profile*" to a "*Process*" that has inputs, outputs, preconditions and effects produced during the execution of a service.
- Classification aspect that describes the type of service. It may precise the service type based on a predefined Service Category.

- Non-functional aspect that makes the difference between similar services. The non-functional parameters can be the QoS, privacy, security, etc.

3.3.1.2 Service Model

The service model contains a sub-class, "*Process*", that indicates to the consumer what he must do to invoke a service. The process could be atomic, composite (composed of atomic processes), or simple (used to provide abstracted views of atomic or composite processes).

While an atomic process has a single interaction between consumer and producer, a composite process has a set of processes linked together by a control flow (sequence, parallelism, etc.) and a data flow that describes how data are interchanged between the processes.

Each process has:

- Participants: the client requesting the process and the server performing the process.
- Parameters: the input, output and local parameters.
- Pre-condition: a condition that the client should satisfy in order to invoke the service. This condition could be an expression, a SWRL rule, or others.
- Result: obtained after the execution of the process. The result has a type and a value.

3.3.1.3 Service Grounding

The service grounding defines how abstract information can be realized by concrete information exchanges. It details the used transport protocol, port number and the message format. Actually, the service grounding class contains only the WSDL sub-class used to exchange SOAP messages.

Figure 3.2 depicts a typical OWL-S interaction Model. A service provider advertises a service described by a service model and a service profile (52). A requester queries for a service. The OWL-S matchmaker verifies if there is an available service that matches the query, if yes, it provides the requestors by the URI of the available services. The requestor then chooses an adequate server and retrieves the service model in order to invoke the service.

3.3.2 IoT Ontologies

The IoT-A project presents a high level abstraction of the IoT domain. It basically consists of a Domain Model, Information model and Communication Model (53). The Domain Model includes a user who can be a Human or an Active Digital Entity. The user interacts with a physical entity and invokes at least one service. A virtual Entity can be associated with a service or/and a resource. The service accesses at least one resource. The resource can be storage, network, or on-device resources (which hosts a device). A device can be an actuator, a sensor or a tag device.

The authors in (54) extended the works done in the IoT-A project by adding semantic description and linking the models to existing domain specific ontologies. The core of the IoT, Information model, consists of Entity, Resource, Service and Device.

3.3.2.1 IoT Information Model

Let us start by the Entity class. An entity can be a Smart entity (cloud smart entity), a Formal Entity, an Information Entity (Sensor data sheet or Sensor Output), a Control entity, or a Physical Entity (Device, facility or person). Figure 3.3 depicts the class hierarchy of the IoT entity class.

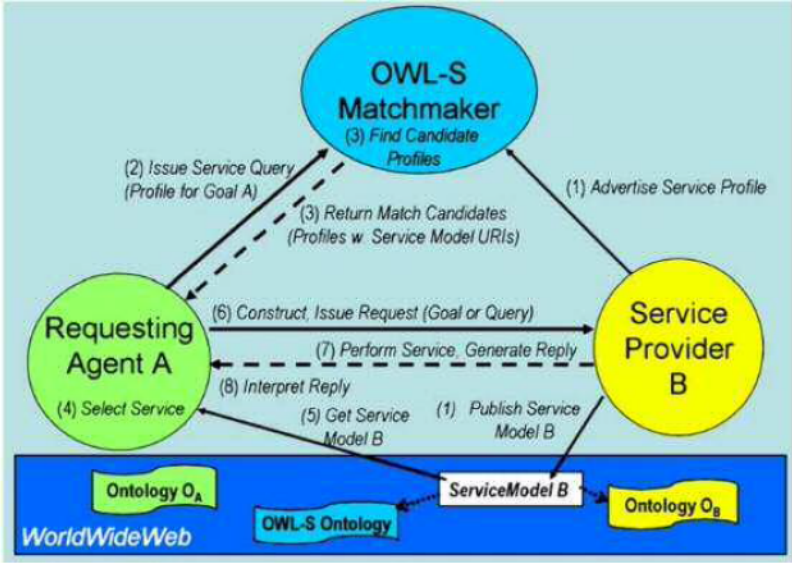

Figure 3.2-Typical OWL-S Semantic Web Service Interaction Model (52)

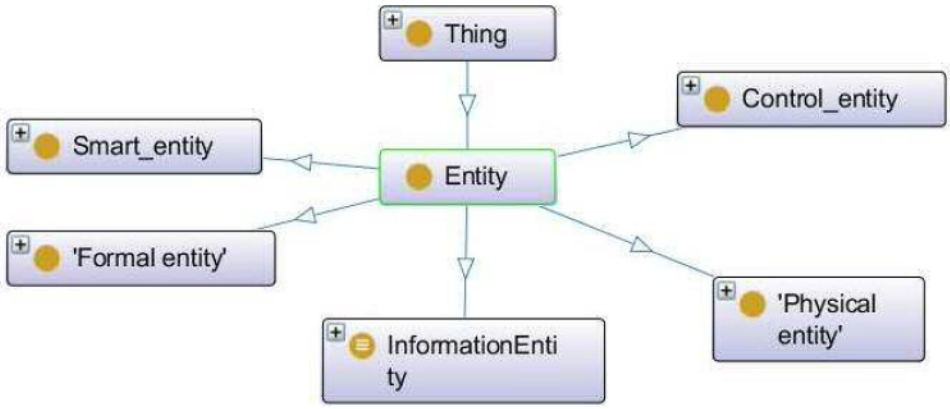

Figure 3.3-Classes Hierarchy of IoT Entity

Moreover, an entity is described by its domain attributes (in general, the observed features), its Location (geo-location or location), its owner (e.g. individual from the "faof"[11] ontology), its global identifier (e.g. an individual from the "dpedia" [12] ontology), and its Temporal features (availability Time and Date).

[11] http://xmlns.com/foaf/spec/

[12] http://dbpedia.org/ontology/

The resources represent the entity in the digital world. Figure 3.4 depicts the Resource Model. The resource has a location (device's location or resource's location), a description, a type (actuator, sensor or tag), and an "*AccessInterface*" which connects the Resource model to the Service model. The "*AccessInterface*" has as type REST, SOA, or RPC. The resources are accessed by the services.

The OWL-S ontology is used as the upper service model, where the Service is the IoT Service, the Service Profile is described in the Entity Model and the Service Grounding is extended to encompass the different IoT messaging techniques. Figure 3.5 depicts the Service Model. The Input Output Precondition Effect (IOPE) of the service profile are described using the domain attributes of the entity model.

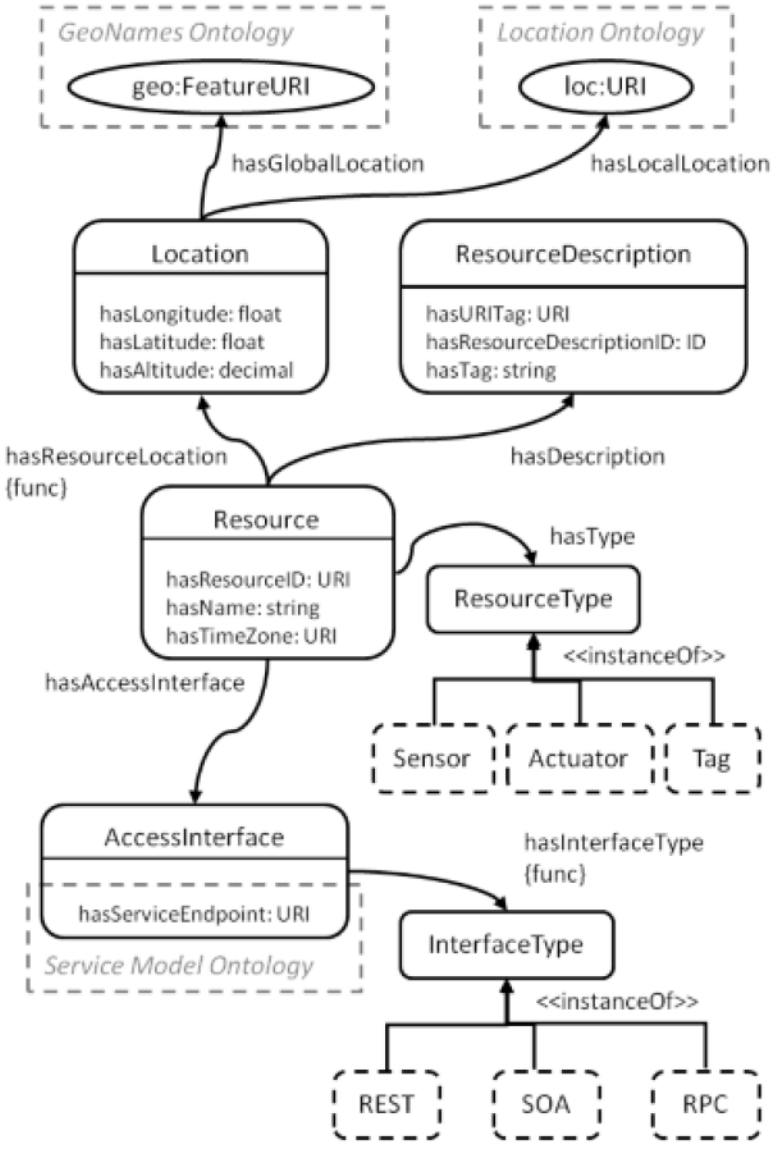

Figure 3.4-IoT Resource Model (54)

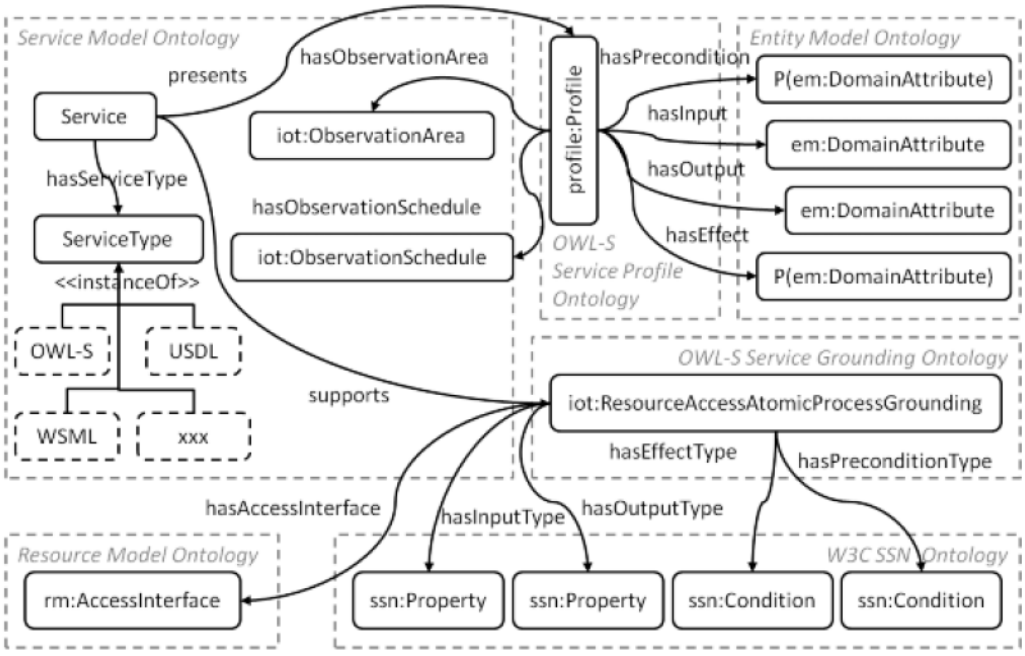

Figure 3.5-IoT Service Model (54)

3.3.2.2 Other IoT Models

More researches have been investigated in order to combine the OWL-S service ontology with the IoT information ontology. Nambi *et al.* (55) proposed a unified semantic knowledge for IoT where resources are readable, recognizable, locatable, addressable and/or controllable via the Internet. The Resource Ontology describes the sensors, actuators, physical objects and composite objects. It is reused from the SSN ontology. The Location Ontology adds the geospatial information to IoT information. It is reused from the GeoNames[13] ontology. The Domain ontology models a specific or a generic domain such as E-health, smart home, etc. Any domain ontology can be imported depending on the IoT application. The Context Domain enables context-awareness and contextual interoperability during service discovery and composition. Aspect-Scale-Context (ASC) is used to model conceptual information. The Policy Ontology defines how to accomplish a service requested by a user. There are three different policy levels: high level policy defining abstract service, concrete policies for specific services, and low level policies defining the execution plan of the services. The Service Ontology is built upon the concept of OWL-S. While Figure 3.6 depicts the service ontology, Figure 3.7 depicts the Location Ontology.

Wang *et al.* (56) introduced a new Semantic Model for IoT architecture composed of seven main modules depicted in Figure 3.8. The IoT resources are defined by extending the SSN ontology to include actuators, servers and gateways. The Observation & Measurement Ontology is reused from the SSN Ontology. The entity of Interest and physical locations represent the object of the physical world.

[13] *www.**geonames**.org/**ontology**/*

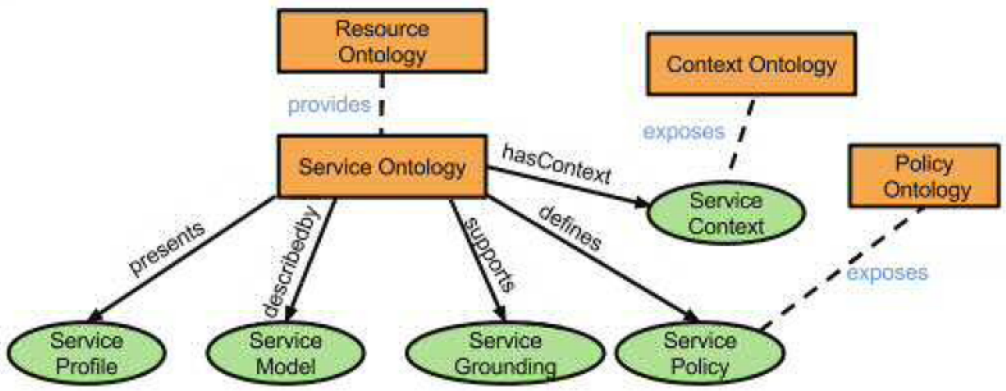

Figure 3.6-Service ontology with important properties (55)

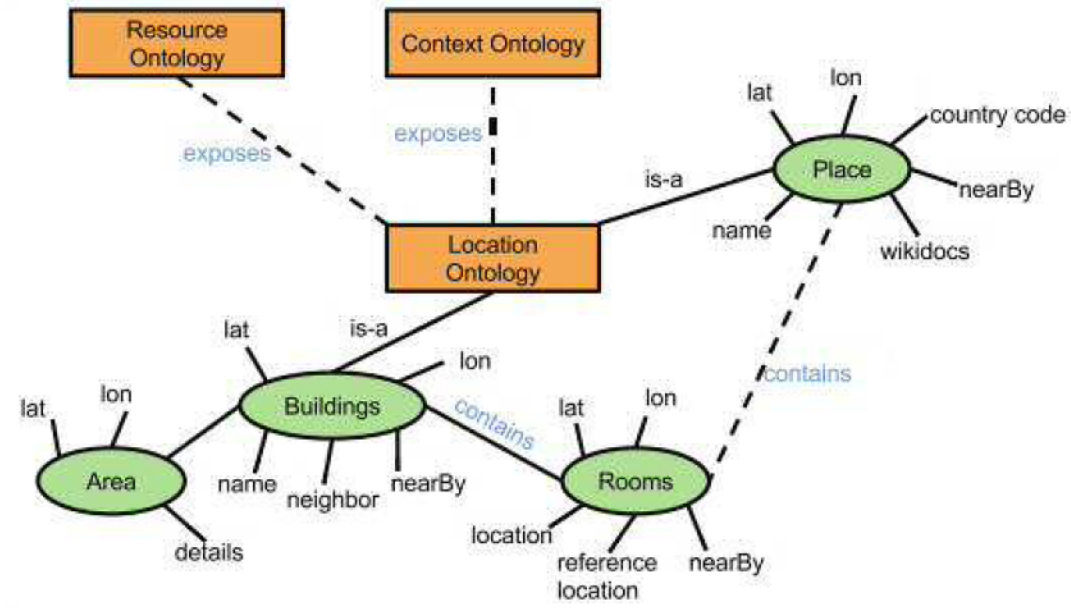

Figure 3.7-Location Ontology (55)

The Deployment Systems and Platforms give the descriptions on how the IoT resources are organized and deployed as well as the system they form. The Quality of Service (QoS) and the Quality of Information (QoI) are essential in the IoT architecture, especially because we are dealing with constrained devices and dynamic environments. The QoS includes the network quality indicators such as jitter, delay, and packet loss, etc. Service Test is used to test functional and non-functional capabilities of IoT services during design and deployment stages. The IoT Service is a subclass of Service in OWL-S ontology. The service type could be SOAP or REST. Each IoT service has an IoT Service test based on IOPE principle.

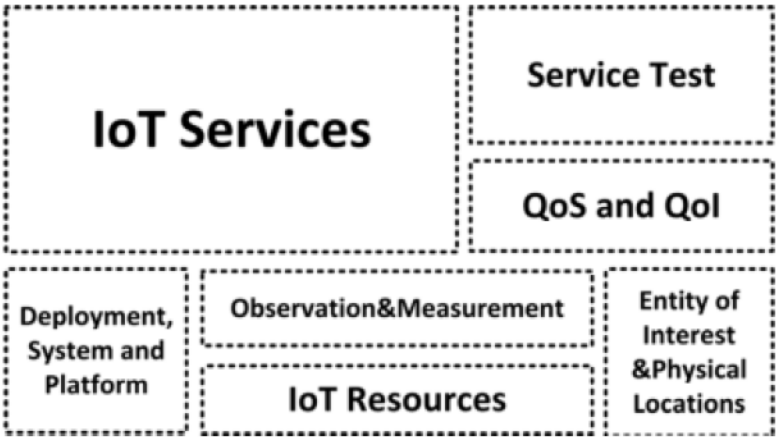

Figure 3.8-Modules in the description Ontology (56)

Hachem *et al.* proposed new ontologies for IoT (57): Device ontology that models the hardware devices in the network, Domain ontology models information about real world physical concepts and forms the service repository, and Estimation ontology that contains information about different estimation models ("linear interpolation", "Kalman filter", "naive Bayesian learning", etc.) in order to help service providers to choose the adequate service accurately.

The device ontology describes the main features of a hardware devices without going into details, because the authors found that the detailed description can overload the decision making process in an IoT networks which contain a huge number of devices. A device can be a sensor, an actuator, a processor that can perform operations, and a composite device composed of minimum two devices.

The physical domain ontology describes the real physical data, as well as it models mathematical formulas and functions to be used when no device can provide the needed services. It models the physical concept, which presents a real world object, or property that can be measured, a physical unit of a measurement, the mathematical data type, which is a set of numbers that can represent a real world property measurement, a formula and a function which is the implementation of the formulas requiring input and output machine data types.

The estimation model will store different mathematical models that make up the mental toolbox carried by expert system designers in fields such as Robotics, Estimation, Sensor Networking, etc. Actually, it encloses the Estimation & Prediction Models used for the automatic discovery, the Association & Correlation Models for data aggregation and correlation, and the Error Models that defines the uncertainty into measurements and actuations.

In summary, some IoT models reuse the SSN ontology to describe the information models (devices, feature of interest and measurements). Others reuse the OWL-S ontology (6) to describe the services offered by the IoT networks. Some initiatives proposed entire new IoT models, where the devices, the discovery and the execution of services are separated into different models. Hachem *et al.* (57) presents an enhancement on the service discovery process in IoT allowing the experts in domain ontologies to define their estimation, prediction and error models for service discovery and caution. But none of these existing ontologies models the part related to IoT devices management like routing, energy consumption, failures discovery, etc.

3.3.3 M2M Ontologies

M2M conceives a general information model (depicted in Figure 3.9) where each physical or virtual M2M object is modelled as "*Thing*" (58). M2M proposed a mapping between ETSI M2M semantic model and the OBIX (Open Building Information exchange) which consists of interlinked objects identified by URIs using XML format (1).

The semantic content is introduced into the ETSI M2M system using:

- Top level Ontologies which are independent of a problem or domain (ex. Time).
- Domain Ontologies that could be given by vertical industries.
- Task and Problem Solving ontologies that define the vocabulary related to certain task, generally provided by service providers.
- Application ontologies that are the most specific ontologies related to certain application.
- SenMESO ontology depicted in Figure 3.10.

Therefore, the semantic information is treated in domain ontologies using rules and reasoning engines. As an example, if we are working in the Health domain and the measurement type is Temperature we can deduce that measured temperature is the body temperature. Moreover, if this temperature is greater than 38, then the person has Flu.

3.4 MyOntoService Ontologies

Based on the existing semantic solutions for services 'description, we conceive our Upper Service Ontology re-using the general structure of OWL-S ontology. Thus, our service ontology is composed of three main sub-ontologies:

- MyOntoServiceProfile.owl: It describes the service and enables the auto-discovery of services.
- MyOntoServiceProcess.owl: It describes how the service is realized based on the IOPE mechanism.
- MyOntoServiceGrouding.owl: It describes how the service can be invoked.

In addition, a general MyOntoService.owl ontology that combines the aforementioned ontologies and includes general and domain SWRL rules is also introduced and defined.

In fact, the reason behind using modular ontology is not only coming from OWL-S (6) but also from the necessity for implementing one part of the ontology in constraint nodes. It becomes pivotal to divide the ontology into sub-ontologies that can be separately treated and reasoned in specific nodes. Going back to the main challenge in SOA, the overhead caused by the processing of input parameters is now not the task of the service itself, but the SPARQL and inference engine that, based on the service process description, can analyse these parameters before invoking the service. Moreover, many applications require customization and personalization, which will be easier using modular ontologies. The remaining part of this Chapter will detail these sub-ontologies.

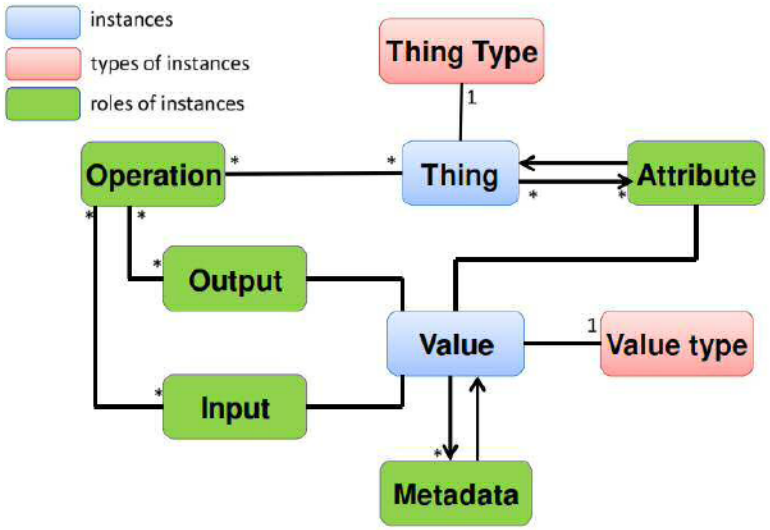

Figure 3.9-M2M Information Model (60)

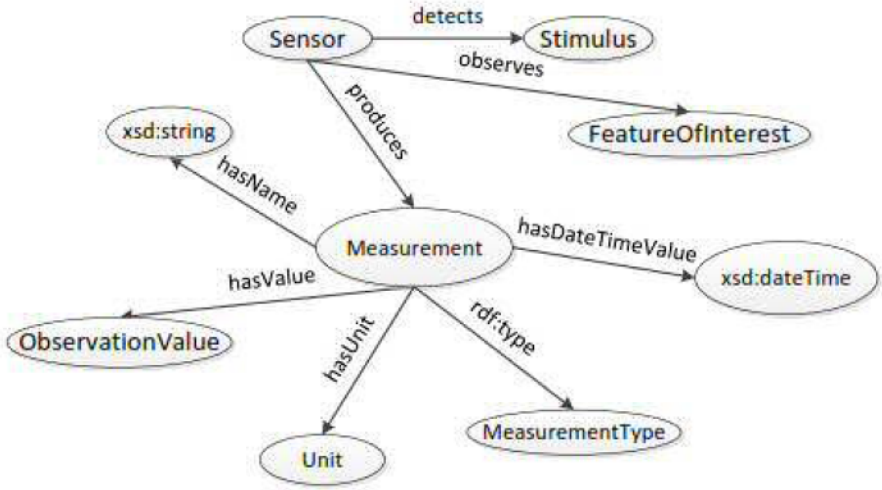

Figure 3.10-SenMESO ontology (1)

3.4.1 MyOntoServiceProfile Ontology

The integration of WSN in WoT imposes that the WSN can be discovered by end users/software. At that end, services offered by WSN should be described in a semantic manner. This is the aim of MyOntoServiceProfile ontology. It fully describes the non-functional characteristics of a service. This information is used to identify the best service that suits the need of the service consumer. Thus, it will help to resolve the problem of bugs or corruption introduced by some services and automatically switches to another service having same functionality.

3.4.1.1 Service Profile Model

First of all, a service profile is described by its name and description (reproduced form OWL-S). Because WSN can cover a wide range of application domain having different requirements, constraints and characteristics, we left the opportunity for the domain experts to classify the services based on their own criteria. This is modelled by the relation *hasServiceClassification* between the *ServiceProfile* class and the domain ontology. In that way, the "*Estimation*" ontology proposed by S. Hachem *et al*. (59) can be integrated into our proposed model.

Because our model in dedicated to WSN, we categorized each service based on a service category that reflects the observed feature. The service category is divided into three sub-categories:

- Body: divided into general and medical. This category groups all services related to human body. It includes general services (acceleration, speed, position, status, location, fall detection, etc.), and medical services (temperature, heart rate, blood pressure, calories burned, etc.).
- Ambient: includes weather services like ambient temperature, pressure, humidity, light intensity, wind speed, etc.
- Object: services related to objects like the velocity, acceleration, temperature, etc.

This categorization is mainly used when a client requests a service. In addition, this semantic description will eliminate the imprecise meaning of certain parameters like body temperature and human temperature. It is modelled by the relation *hasServiceCategory*. In a WSN, services are basically:

- data collected from sensors,
- commands that should be executed by actuators,
- alerts that inform the user about urgent cases, and
- notifications that notify the user, or that reminds the user about an event that he/she assigned.

Knowing the type of service permits the users to choose which service he/she wants to use. Let's say a user is interested in monitoring his blood pressure. He can choose a monitoring service where:

- the data are periodically collected and displayed,
- or a notification is sent when his blood pressure is changing,
- or an alert is sent when his blood pressure is abnormal.

Thus, the *ServiceCategory* class is used to differ from two services related to the same observed data. In addition, this class is very effective for service provider/developer because it allows them to use the adequate service category based on their users' needs. If the client is requesting a fall detection service that sends him/her a notification, when the service discovery agent is searching for available services, the system will search for a service which has as service category fall detection and service type notification.

Dealing with various types of data (periodic, urgent, time sensitive, etc.) pushed us to integrate the concept of:

- QoS (Quality of Service), indicating the quality of the provided service,
- and QoI (Quality of Information), indicating the quality of the information delivered by the service,

in our proposed service model. Re-using the QoS and QoI classes of the IoT (54) model, the service profile in our service model has the QoS and the QoI as parameters. QoI parameters are calculated from the QoS parameters using certain formulas that are also saved in our proposed ontology. All these parameters permit to choose the adequate service between different services having same category and type. Let's say we have two sensors gathering the body temperature, but the accuracy of one differs from the other. In this case, the QoI will permit to select the more accurate thermometer to retrieve the body temperature. Furthermore, we added the service constraints (security, legal, time validity, etc.). These constraints are modelled by the *Constraints* class in MyOntoSens ontology. Finally, like for OWL-S and in our proposed model, the service profile is associated to a service process. In OWL-S, a service profile has contact information. Or, in our ontology, each WSN has a contact. Thus, we didn't add the contact information to the service profile ontology, but we instead linked the WSN ontology to the service ontology by the following relation: WSN *isEngaged* in Service. Accordingly, the contact information can be deduced based on a SWRL rule explained in section 3.4.4. Figure 3.11 depicts the class diagram of "MyOntoServiceProfile" ontology.

3.4.1.2 Service Profile Ontology Description

This section details the object properties and data properties of "MyOntoServiceProfile" ontology. The full description of the ontology is given in Annex A.5.

Object Properties:

Table 3.1-MyOntoServiceProfile Object Properties

Object property	Domain	Range	Additional description
hasCalculationMethod	QoS	QoI	
hasServiceCategory	ServiceProfile	ServiceCategory	
hasServiceClassification	ServiceProfile	Any imported ontology	
hasServiceParameter	ServiceProfile	ServiceParameter	
hasQoS	ServiceProfile	QoS	subProperty of *hasServiceParameter*
hasQoI	ServiceProfile	QoI	subProperty of *hasServiceParameter*
hasConstraints	ServiceProfile	ServiceComstraints	subProperty of *hasServiceParameter*
hasServiceProcess	ServiceProfile	MyOntoServiceProcess: ServiceProcess	
hasServiceType	ServiceProfile	ServiceType	

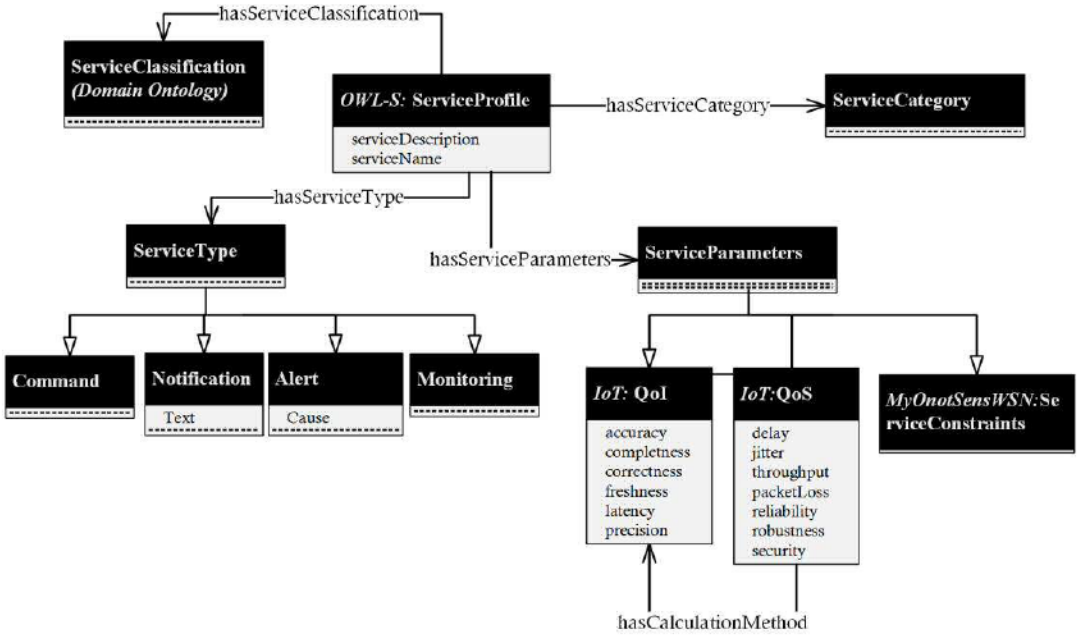

Figure 3.11-MyOntoServiceProfile Class Diagram

Data Properties:

Table 3.2-MyOntoServiceProfile data properties

Data Property	Domain	Range	Additional property
Accuracy	QoI		
Completeness	QoI		
correctness	QoI		
Freshness	QoI		
Latency	QoI		
Precision	QoI		
Delay	QoS		
Jitter	QoS		
Throughput	QoS		
packetLoss	QoS		
Reliability	QoS		
Robustness	QoS		
Security	QoS		
serviceName	ServiceProfile	anyURI	Functional
serviceDescription	ServiceProfile	String	
Cause	Alert	String	
Code	ServiceType		
CommandDescr	Command	String	
Text	Notification	String	

3.4.2 MyOntoServiceProcess Ontology

The main goal of this sub-ontology is to describe how the service is executed and how a client can interact with the service. Same as in OWL-S, it uses the IOPE principle. The same service profile can have many service process models. For example, if we are providing a fall detection service, it can be based on the calculation of the linear acceleration or based on the angular and linear acceleration. In this case, two different process models are combined to the same service profile. Moreover, when a user requests a service, he/she should know what the required input parameters are and pre-conditions or not. This can be deduced from the service process model. Note that the user can be a software, application, service or end user. The detail of the service process model is given in the next section.

3.4.2.1 Service Process Model

OWL-S defines process as an atomic process or a composite process expanded into one or more simple processes. We reproduced the same idea, but we transposed the use case to a WSN.

Basically in WSN, the trivial service is the service providing the data collected by a sensor. In MyOntoSensProcess ontology, we described that each sensor is used in a process. Thus, this process can be considered as the trivial service offered by the WSN (like temperature, acceleration, blood pressure, etc.). Consequently, each process is an atomic service process in MyOntoServiceProcess ontology.

To enhance the semantic description of the composite processing we added the *ProcessRelation* class indicating the relation between two, or more, simple processes expanding from the same composite process. The relation could be an order (first, before, etc.), parallel, or a condition between two simple processes.

Re-using the "*Method*" class from OWL-S (6) and inspiring by the Estimation ontology proposed by S.Hachem *et al*., we added the relation *hasCalculationMethod* between the service process class and the method class. Our contribution is in adding sub-classes to the class method where a method can be an aggregation (maximum, minimum, etc.), or a specific formula (least square, etc.).

As mentioned previously, the service process has input parameters, output parameters, precondition(s), and result (s). We added the *Optional* property to the parameter class as Boolean. We considered that each service process can have:

- Input parameters (same as OWL-S), e.g. the patient being monitored.
- Output parameters (same as OWL-S), e.g. the observed value.
- Pre-condition (same as OWL-S), e.g. the user permission to access the monitored data.
- Effect. In our case, the effect can be a display, notification, sound effect, command, action, etc.
- Post-condition, a condition to be satisfied when the process is executed.

To clarify the idea, let us consider that patient relatives can monitor the patient's heart rate. The service is the monitoring of the heart rate. The inputs are the time of the requested service and the requestor identity to verify the permission (in case of using a smartphone for monitoring, it can be his/her email), the precondition is that the heart rate sensor is enabled in order to retrieve data, the effect is a display on the mobile's screen of the relative and the post condition could be a confirmation from the patient that this relative can check his/her heart rate although he/she already has the permission.

Likewise, OWL-S defined for each process participant using the object property "*hasParticipant*". This object property has two sub-properties: "*hasClient*" and "*performedBy*". In general in WSN, an application has end users or software as client and is performed by a node. But what is really essential in WSN is also to know the producer of the data. Referring to the IoT ontology, we utilized the concept of Entity but in different manner. In our proposed model, an entity can be a physical entity (person, node or object), or virtual entity (software, social person or application). Thus, we added the object property "*hasParticipant*" between the service process class and the Entity class. This object property has two sub-object properties: *hasConsumer* and *hasProducer*. While Figure 3.12 depicts the service process class diagram, Figure 3.13 depicts the sub-classes of the Entity class.

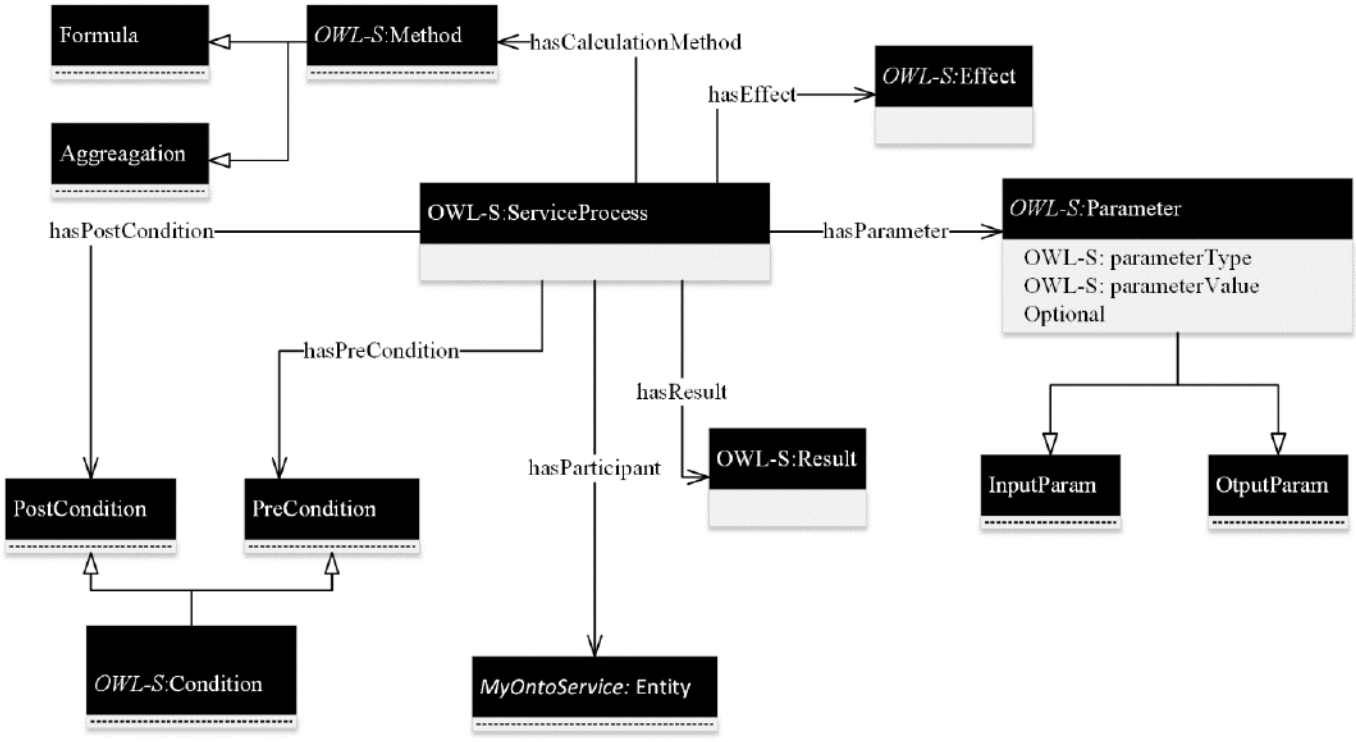

Figure 3.12-Service Process Class Diagram

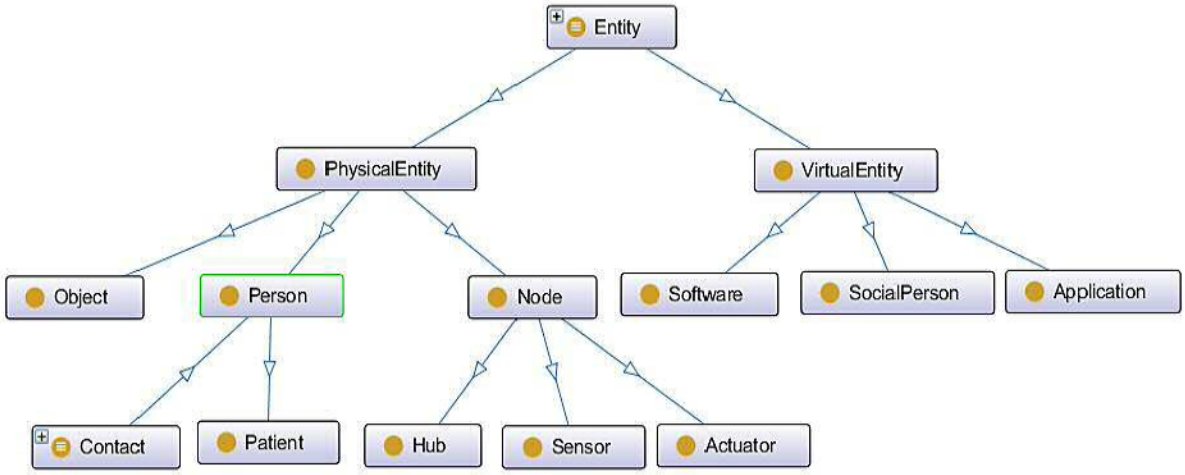

Figure 3.13-Entity sub-classes

3.4.2.2 Service Process Ontology Description

The "Atomic" class is equivalent to "Process" class in MyOntoSensProcess ontology. In addition, the "ServiceProcess" class is an abstract class that can be a composite, simple or atomic class.

Object Properties:

Table 3.3-MyOntoServiceProcess object properties

Object property	Domain	Range	Additional Description
hasPostCondition	QoS	QoI	
hasPreCondition			
collapses	Simple	Composite	Inverse Of: expandsTo
expandsTo	Composite	Simple	Inverse Of: collapses
hasCalculationMethod	ServiceProcess	Method	
hasParameter	ServiceProcess	Parameter	
hasInputParam	ServiceProcess	InputParam	Sub-property Of: hasParameter
hasOutputParam	ServiceProcess	OutputParam	Sub-property Of: hasParameter
hasProcessRelation	Composite	ProcessRelation	
hasResult	ServiceProcess	Result	
hasEffect	ServiceProcess	Effect	
isAssociatedWith	ServiceProcess	(Domain Ontology)	
isRelatedTo	ProcessRelation	Simple	
isAfter			Sub-Propety Of: isRelatedTo
isBefore			Sub-Propety Of: isRelatedTo
isInParallel			Sub-Propety Of: isRelatedTo
hasParticipant	ServiceProcess	*MyOntoService:* Entity	
hasProducer			Sub-Property Of: hasParticipant
hasConsumer			Sub-Property Of: hasParticipant

Data Properties:

Table 3.4-MyOntoServiceProcess data properties

Data Property	Domain	Range
Optional	Parameter	boolean
parameterType	Parameter	
parameterValue	Parameter	
RelationType	ProcessRelation	string

3.4.3 MyOntoServiceGrouding Ontology

The service grounding ontology determines how the service is invoked and how to map semantic data to required data for transmission. This ontology permits to find the service based on the technology enabled on the client side.

As defined in OWL-S, each service grounding has a determined port number, reference and protocol version. OWL-S restricted the grounding sub classes to the SOAP technique for services. It offered the possibility to add REST techniques. With the evolution of web services techniques new protocols appeared for data transmission. We can cite here, MQTT (60) or CoAP (31) dedicated for constraint devices, the JSON and JSON-LD (61) for non-XML data transmission, HL7 (62) messages for biomedical data, Bluetooth LE messages for Bluetooth devices (45), etc. For that reason, we extended the sub-class hierarchy of the grounding class as depicted in Figure 3.16 and we just link the service profile to two classes: the *MessageElements* and *MessageOptions* classes as presented in Figure 3.14. These classes describe the optional/required fields with its characteristics in order to form a message based on one of the service grounding class. We left the mapping method to be described in our overall system based on multi-agents architecture (see Chapter 4) where each service grounding is mapped using a mapping agent responsible of retrieving semantic data from the ontology to form the message and vice-versa.

3.4.3.1 Service Grounding ontology description

Object Properties:

Table 3.5-MyOntoGroudingServie object properties

Object property	Domain	Range
hasMessageElements	ServiceGrouding	MessageElements
hasMessageOptions	ServiceGrouding	MessageOption

Data Properties:

Table 3.6-MyOntoGroudingServie data properties

Data Property	Domain	Range
FieldDescription	MessageElements or MessageOption	String
FieldLenght	MessageElements or MessageOption	Integer
FieldName	MessageElements or MessageOption	String
FieldOrder	MessageElements or MessageOption	Integer
FieldType	MessageElements or MessageOption	
Required	MessageElements or MessageOption	Boolean
Port	ServiceGrouding	Int
Reference	ServiceGrouding	anyURI
Version	ServiceGrouding	String

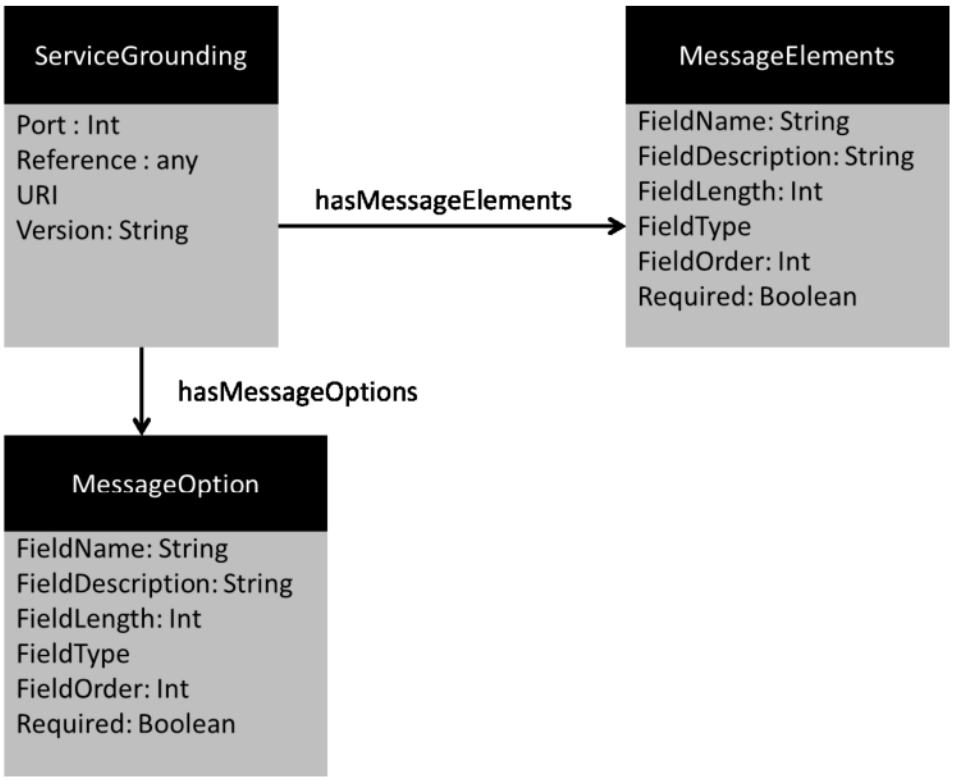

Figure 3.14-MyOntoServiceGrouding class diagram

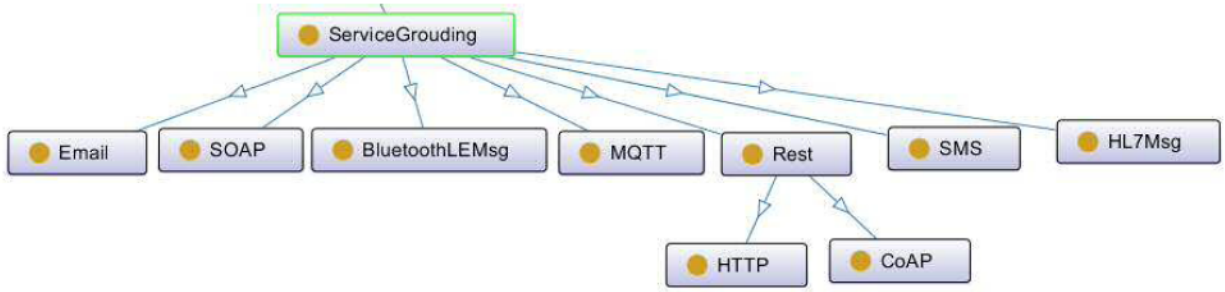

Figure 3.15-MyOntoServiceGronding sub-classes

3.4.4 MyOntoService Ontology

MyOntoService ontology is the global service ontology that directly imported the three previously mentioned ontologies. It contains the Service class described by the service's name and service description (same as in OWL-S). In addition, it contains the object properties that link the three ontologies (MyOntoServiceProfile, MyOntoServiceProcess and MyOntoServiceGrouding). Figure 3.16 depicts the class diagram of the MyOntoService ontology.

Although a service may be available for the end users, the owner of a service can choose to disable the service. That's why the object property *enable* is added between the Person class and the Service class.

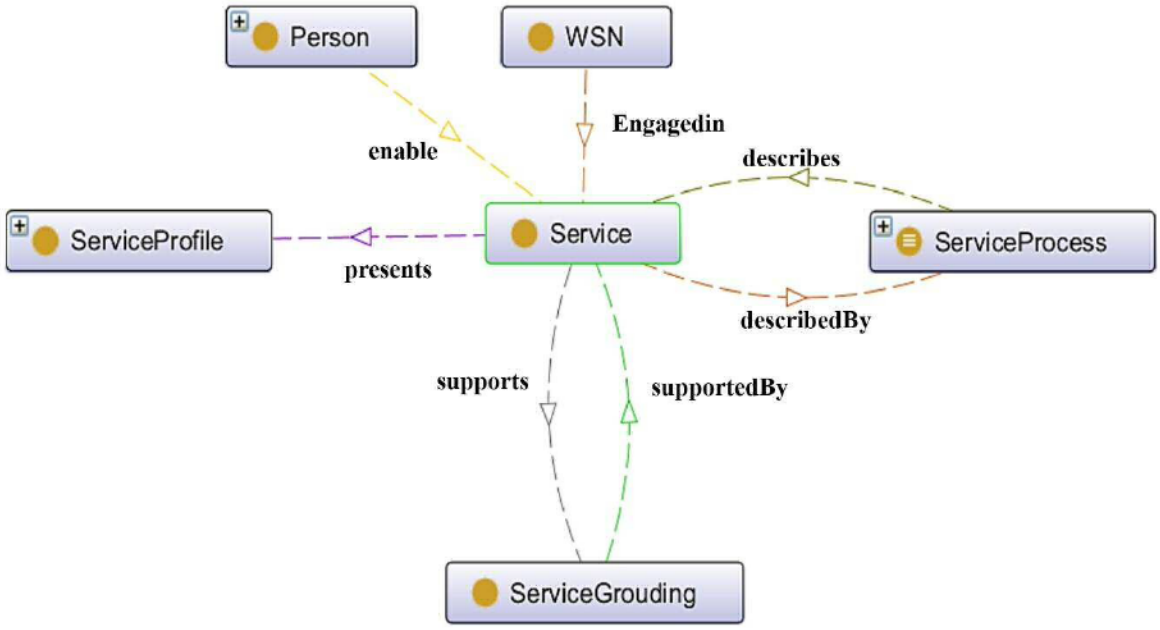

Figure 3.16-Service Classes schema

To give more expressiveness to the proposed service ontology, SWRL rules are added. Firstly, the rules related to the trivial service offered by each WSN which is the process modelling the observed value of a sensor, were added. The corresponding set of rules is the following:

1- Each process is considered as a service

(Rule 3.1) Process(?P) -> Service(?P)

2- Each atomic process has as producer the sensor

(Rule 3.2) Sensor(?S), Atomic(?a), usedFor(?S, ?a) -> hasProducer(?a, ?S)

3- The service ID of the process is the same as the process ID.

(Rule 3.3) Process(?P), ServiceProfile(?P), ProcessID(?P, ?PID) -> ServiceID(?P, ?PID)

4- If a process has constraints the service profile should have the same constraints

(Rule 3.4-a) Process(?P), Security(?L), ServiceProfile(?P), hasConstraint(?D, ?L), hasData(?P, ?D) -> hasConstraint(?P, ?L)

(Rule 3.4-b) Legal(?L), Process(?P), ServiceProfile(?P), hasConstraint(?D, ?L), hasData(?P, ?D) -> hasConstraint(?P, ?L)

Secondly, when a sensor in a WSN is used in a process, this WSN is surely engaged in this service (equivalent to the process). This leads to the following rule:

(Rule 3.5) ServiceProfile(?P), BelongToCluster(?S, ?C), usedFor(?S, ?P), belongTo(?C, ?W) -> Engagedin(?W, ?P)

Thirdly, if two simple processes are related by a sequence relation, if one simple server is before the other, assuredly the second one is after. This leads to the following rule:

(Rule 3.6) Simple(?P1), Simple(?P2), isAfter(?P1, ?P2) -> isBefore(?P2, ?P1)

Finally, (Rule 3.7) is added for the *isRelativeof* property.

(Rule 3.7) isRelativeof(?P1, ?P2) -> isRelativeof(?P2, ?P1)

Note that more rules can be added depending on the domain of application. A fully detailed example is given in the next section.

3.5 Pre-Validation of the Upper Service Ontology

To pre-validate the overall ontology, MyOntoSens and MyOntoService ontologies were imported to a common ontology and the Pellet reasoner is run. The ontology is well classified without any inconsistency by the reasoner. The next step is to choose a scenario to validate the inferred information.

Being part of the ETSI TC SmartBAN technical committee and related projects, we focused our work on the integration of our model in biomedical scenarios and existing biomedical open data model. First of all, we verified that our upper service layer can be used for semantic EHR systems.

In fact, the semantic meaning in EHR data was integrated by defining Archetypes. An *EHR archetype* is an *agreed*, *formal* and *interoperable* specification of the data and their interrelationships that must or may be logically persevered within an EHR record for a clinical observation, evaluation instruction or action (63). The UCL Ocean Informatics (64)conceived in 2008 an advanced Information model for openEHR. This model defines a logical EHR

information architecture, rather than just architecture, for communication of EHR extracts or documents between EHR systems. In this model, for each patient, an EHR object (as root) is created. All the data related to this patient is in the Composition class which are the data container in the openEHR information model. For each composition, the language, territory, category and composer (who created the data) are added. One or more events can be associated to a composition. Each event is described by its start date, end date, location and participants. The events could be a problem list, a family history, a care plan, etc. The basic information of the patient (DoB, sex, height, etc.), in addition to other medical data (allergy, status (e.g. if he has cancer)) are saved in the model. The problem list indicates the result of the observation analysed by the clinical staff. This information model can be integrated in our proposed upper service model as follow:

- The patient is described in MyOntoSensWSN ontology in the Patient class.
- Data are described in MyOntoSensProcess ontology in the data and measurement classes.
- Each composition can be considered as a service described in MyOntoServiceProfile ontology. The service category could be a problem list, medication, therapist precaution, family history, etc.
- Such compositions are produced (e.g. how a clinician decided a problem list for a patient) can be defined in MyOntoServiceProcess ontology, in addition to the participants that already exist in our proposed upper service model.
- Access to data (HL7 messages) can be described in the MyOntoServiceGrouding ontology.

Thus, this upper service layer can be used to enhance the openEHR information model by describing the sensors and equipment used in order to capture data. Moreover, new SWRL rules can be added to enhance the decision making for patients. For example, information from family history combined with patient status and laboratory result can conduct to confirm if he/she is suffering from certain disease or if he/she has to take certain precautions. So, in that case the precautions can be viewed as effect of the service process.

After the validation of the integration of our upper service ontology with openEHR, we pre-validate the ontology in a particular biomedical use case. Since the percentage of elderly people living alone is increasing, the need to design an autonomous system that is able to monitor their health conditions and provide healthcare services is rising. These systems should provide autonomous service discovery in order to help them to live independently and safely. This kind of system seems to provide a typical scenario to test our proposed service ontology. A smart home dedicated for monitoring elderly is described in section 3.5.1.

3.5.1 Smart Home System description

In general, RPM (Remote Patient Monitoring) technologies, also called homecare tele-health, have the potential to help patients that are challenged by chronic and acute illnesses and/or injuries. The system consists of sensors (biomedical sensors, location sensors, intrusion sensors, movement detectors, etc.), which continuously collect the data and send it to a local server (for example, the home server or the user's Smartphone). The data can be treated within the local server, and/or sent to remote healthcare and monitoring servers. Basically, the system provides

monitoring, alarm generation, reminders, and remote advices services. In the context of this thesis, we considered the smart home system illustrated in Figure 3.17.

The patient is wearing an H7Polar heart sensor, a temperature sensor and has his/her own Smartphone. Three-axial accelerometer built in the smart phone is used to detect the fall of the elder. Our proposed solution takes into consideration that a fall initially provokes a sudden and significant decrease in the acceleration amplitude. After this "free-fall-period", the acceleration experiences an abrupt spike as soon as the body hits the floor. Consequently, if the acceleration crosses a lower and an upper threshold during a certain observation time window, a fall is suspected. A fall must start with a short free fall period. This causes the acceleration's amplitude to drop significantly below the 1 g (gravitational acceleration) threshold. This represents the period of time when the actual fall is taking place. Then, the fall must stop and it causes a spike in the graph. Thus, if the acceleration's amplitude crosses the lower and upper thresholds in the set duration period, a fall is suspected. Moreover, the system should detect four kinds of static posture: standing, bending, sitting and lying. If the acceleration's amplitude is equal to the gravitational force, then the posture is static. If $|a_y|$ is greater than 9m/s^2, we can deduce that the person is standing. Otherwise, if $|a_z|$ is greater than 9m/s^2, the person is lying. Finally, if $|a_x|$, $|a_y|$ and $|a_z|$ are between 4 and 6, the patient is sitting. These thresholds are chosen based on personal testing where the smartphone was placed vertically on the chest of the patient. Although these threshold will not give high accuracy for the status, but our main aim is to validate MyOntoService ontology not to provide the ideal solution for patient's posture detection including the fall.

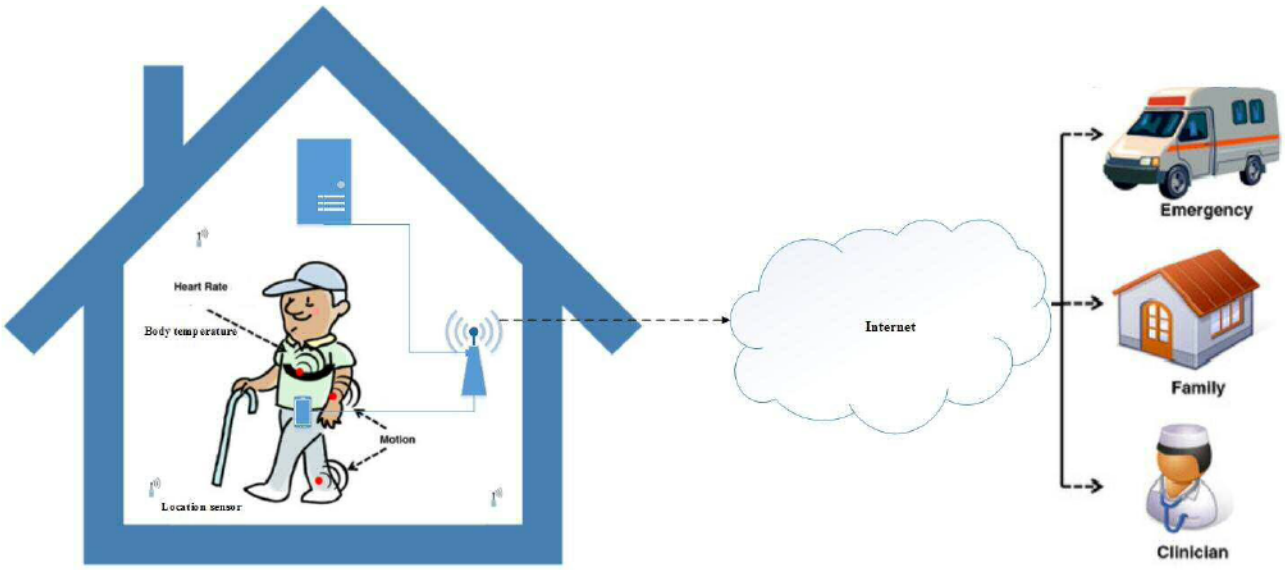

Figure 3.17-Smart Home Architecture

Locating the patient in his home is primordial in such system. Therefore, Bluetooth beacons are used to determine the location of the patient. A local server will collect all the data and create the individuals in the ontology. The system can monitor more than one patient living in the same home and using only one local server. Note that the details of the individuals' creation process are given in Chapter 4. In the next section, a description of the individuals is depicted.

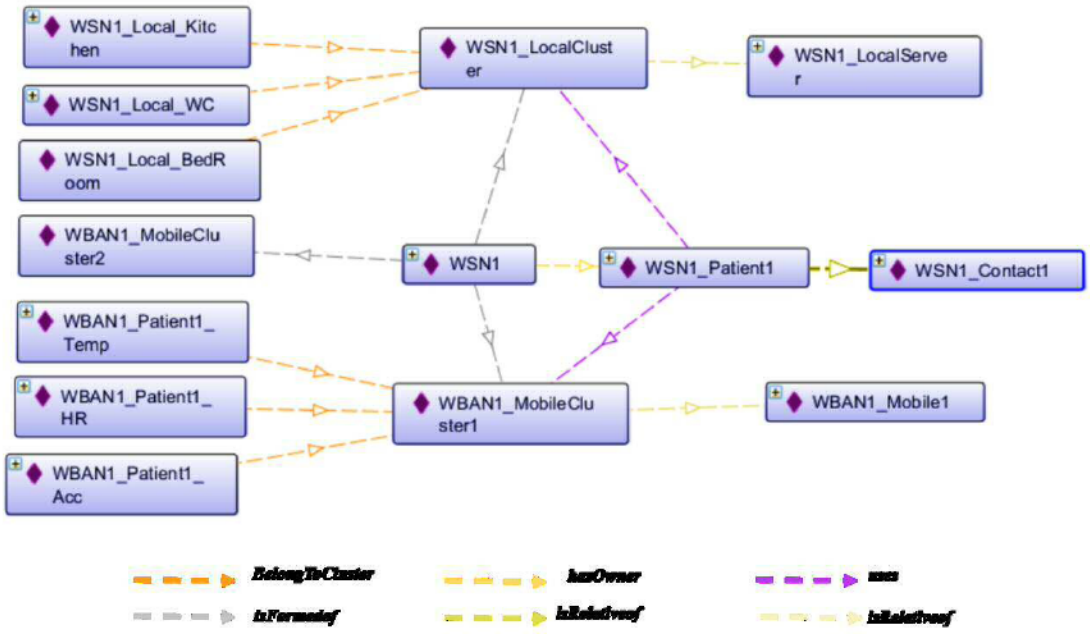

Figure 3.18-Reliationships between WSN, Nodes, Clusters &Persons individuals

3.5.2 Smart Home semantic individuals

Each local home is modelled as a WSN identified by the MAC address of the local server and described by the deployment time. In particular, this WSN has as application domain Smart Home Monitoring. The WSN has as owner the patient and has contact any relatives of the patient. The patient is described by his email, year of birth, gender and name. The WSN has as location the GPS coordinates captured from the mobile phone of the user. As an example, a relative is added to the ontology where this relative is the child of the monitored patient (WSN1_Patient1 "*hasChild*" WSN1_Contact1).

In fact, emails are unique for each individual. And because our intension is to create a mobile application for this use case (in all the mobile phone, each user can be identified by an email account), we considered the email as identifier for the person. Thus, we created Rule 3.8 as follow:

(Rule 3.8) Person(?P), email(?P, ?email) -> UserID(?P, ?email)

The WSN is composed of the:

- WBAN1_MobileCluster1 identified by the mobile phone's MAC address. This cluster is formed of, on one hand, the smart phone and the built sensors (accelerometer, ambient temperature), on the other hand, the H7 polar sensor. Consequently, WBAN1_MobileCluster1 will elect the mobile phone as hub.
- WSN1_LocalCluster identified by the MAC address of the local server. This cluster is formed of three Bluetooth localization sensors and the local server which plays the role of the hub.

Because more than one patient can own the same WSN, the object property "*use*" between patient and cluster should be added. So in our scenario, WSN1_Patient1 uses

WBAN1_MobileCluster1 and WSN1_Patient1 uses WSN1_LocalCluster. WSN1_Patient2 uses WBAN1_MobileCluster2 and WSN1_Patient2 uses WSN1_LocalCluster.

Figure 3.18 depicts the relationships between the following classes: WSN, Nodes, Cluster and Person.

Let us detail now the properties of the nodes (sensors and Hub). Note that the full description of the individuals is presented in Annex A.9.

The Local server, as well as the mobile phone, has a Bluetooth interface and a WIFI interface. The WIFI interface of the local server is the gateway. Thus, any external user can access this home system via this interface. The Heart rate sensor has as type H7Polar sensor, and as position "chest".

As defined in MyOntoSens ontology, each sensor should be used for a process. Each process has data and each data has measurements. Three localization sensors (Kitchen, Bedroom, WC) have been adopted. Each sensor is placed in a room indicating the presence or not of the patient within the room. Furthermore, the object property *measuredFor* is added between the measurements and the patient. The processes are summarized in Table 3.7.

Table 3.7-Process individuals

Process	Data	Data Type
Linear Acceleration	ax, ay, az, and a	Real
Temperature	Temp	Real
IndoorLocalization	Kitchen, WC, Bedroom	Boolean
HR	HRData	Int

The module of the linear acceleration "a" is deduced from Rule 3.9

(Rule 3.9) Sensor(?S), isMeasuredBy(?mes, ?S), usedFor(?S, LinearAcceleration), hasData(LinearAcceleration, a), hasData(LinearAcceleration, ax), hasData(LinearAcceleration, ay), hasData(LinearAcceleration, az), hasMesurement(a, ?mes), hasMesurement(ax, ?mesax), hasMesurement(ay, ?mesay), hasMesurement(az, ?mesaz), TimeStamp(?mesax, ?t1), TimeStamp(?mesay, ?t1), TimeStamp(?mesaz, ?t1), value(?mesax, ?valax), value(?mesay, ?valay), value(?mesaz, ?valaz), add(?res, ?newax, ?neway, ?newaz), multiply(?newax, ?valax, ?valax), multiply(?neway, ?valay, ?valay), multiply(?newaz, ?valaz, ?valaz), pow(?newres, ?res, 0.5) -> TimeStamp(?mes, ?t1), value(?mes, ?newres)

The step dedicated for node discovery has been carried out. The next step to be performed is the creation of individuals for the automatic service discovery.

Dealing with a system dedicated for smart home monitoring, some additional domain based rules can be added. First or all, to access any service provided by this system, the service consumer should precise the patient and should have the permission to access the data. Thus, all the services must have the Patient ID as input and user permission as pre-condition (Rules 3.10).

(Rule 3.10-a) Atomic(?Service), WSN(?W), Engagedin(?W, ?Service), hasOwner(?W, ?P) -> hasInputParam(?Service, PatientID), Optional(PatientID, false)

(Rule 3.10-b) WSN(?W), ServiceProcess(?service), hasOwner(?W, ?Patient), Engagedin(?W, ?service) -> hasPrecondition(?service, UserPermission)

Obviously, each atomic process has as result the measurements captured by the corresponding sensor. Therefore, Rule 3.11.c is created.

(Rule 3.10-c) Sensor(?S), WSN(?W), Atomic(?Service), BelongToCluster(?S, ?C), measures(?S, ?mes), usedFor(?S, ?Service), belongTo(?C, ?W), Engagedin(?W, ?Service) -> hasResult(?Service, ?mes)

Moreover, the categorization of the services is done using the SWRL rules described in Table 3.8.

Table 3.8-Service categorization by SWRL rules

Service	**Service Category**	**SWRL rule**
IndoorLocalization	General: Tracking	ServiceProfile(IndoorLocalization) -> hasServiceCategory(IndoorLocalization, Tracking)
LinearAcceleration	General: MovementAcceleration	ServiceProfile(LinearAcceleration) -> hasServiceCategory(LinearAcceleration, MovementAcceleration)
Temperature	Medical: BodyTemperature	ServiceProfile(Temperature) -> hasServiceCategory(Temperature, BodyTemperature)
HR	Medical: HeartRate	ServiceProfile(HR) -> hasServiceCategory(HR,HeratRate)
SmartHomeFallDetection	General:FallDetection	ServiceProfile(SmartHomeFallDetection) -> hasServiceCategory(SmartHomeFallDetection, FallDetection)

Due to the uses of the previously mentioned rules, all atomic services are automatically discovered, categorized and described by their input parameters, output parameters, producers, and their pre-condition.

Targeting to provide more complex services, the creation of simple and complex service processes is essential. When monitoring the heart rate, in addition to the atomic process, a simple process which sends notification in case the maximum heart rate exceeds the normal, or a simple process which calculates the average heart rate, could be provided. Thus, for the same service profile "heart rate", two simple processes have been created: average heart rate (AvgHR) and maximum heart rate (MaxHR). All the object properties of these simple processes are

created using the rules presented hereafter (Rule 3.11). The main reason behind the creation of these rules is to enable the automatic discovery of the simple processes without the need to enter additional data from the user or the service provider.

(Rule 3.11- a) ServiceProcess(MaxHR) -> hasCalculationMethod(MaxHR, Max), hasInputParam(MaxHR, PatientID), hasOutputParam(MaxHR, MaxHRValue), hasPrecondition(MaxHR, UserPermission), hasEffect (MaxHR,CallNotification)`

(Rule 3.11- b) ServiceProcess(AvgHR) -> hasPrecondition(AvgHR, UserPermission), hasCalculationMethod(AvgHR, Average), hasInputParam(AvgHR, PatientID), hasOutputParam(AvgHR, AverageMesValue)

The same idea can be applied for any simple process deduced for an atomic process where it is beneficent to calculate the average value and the maximum value.

Let us pass now to the main aim of SmartHome monitoring specific application, the determination the patient's Status. The acceleration measurements are used to determine if the patient is sitting, lying, standing, or falling. Moreover, he/she can be in danger (DangerHR) if his/her heart rate exceeded the normal value. For that reason, a new class *Status* has been added. The corresponding object property *hasStatus* between *Person* and *Status* has been created. We calculated the status considering that the smartphone is on the waist of the user vertically. Based on experimental tests, the status of a person is deduced from the rules 3.12:

Standing *(Rule 3.12-a) WSN(?W), ComposedOf(?C, ?S), usedFor(?S, LinearAcceleration), hasData(LinearAcceleration, a), hasData(LinearAcceleration, ay), hasMesurement(a, ?mesa), hasMesurement(ay, ?mesay), hasOwner(?W, ?P), isFormedof(?W, ?C), Engagedin(?W, LinearAcceleration), uses(?P, ?C), TimeStamp(?mesa, ?t1), TimeStamp(?mesay, ?t1), value(?mesa, ?vala), value(?mesay, ?valay), abs(?vala, ?vala), abs(?valay, ?valay), greaterThan(?vala, 9), greaterThan(?valay, 8.5), lessThan(?vala, 10.5) -> hasStatus(?P, Standing)*

Lying *(Rule 3.12-b) WSN(?W), ComposedOf(?C, ?S), usedFor(?S, LinearAcceleration), hasData(LinearAcceleration, a), hasData(LinearAcceleration, ax), hasMesurement(a, ?mesa), hasMesurement(ay, ?mesay), hasOwner(?W, ?P), isFormedof(?W, ?C), Engagedin(?W, LinearAcceleration), uses(?P, ?C), TimeStamp(?mesa, ?t1), TimeStamp(?mesax, ?t1), value(?mesa, ?vala), value(?mesax, ?valax), abs(?vala, ?vala), abs(?valax, ?valax), greaterThan(?vala, 9), greaterThan(?valax, 8.5), lessThan(?vala, 10.5) -> hasStatus(?P, Lying)*

Falling *(Rule 3.12-c) WSN(?W), ComposedOf(?C, ?S), usedFor(?S, LinearAcceleration), hasData(LinearAcceleration, a), hasData(LinearAcceleration, az), hasMesurement(a, ?mesa1), hasMesurement(a, ?mesa2), hasOwner(?W, ?P), isFormedof(?W, ?C), uses(?P, ?C), Engagedin(?W, LinearAcceleration), TimeStamp(?mesa1, ?t1), TimeStamp(?mesa2, ?t2), value(?mesa1, ?val1), value(?mesa2, ?val2), greaterThan(?val2, 9),lessThan(?val1,1),subtractDayTimeDurations (?t3,?t2,?t1), greaterThan(?t3,0), lessThan(?t3,1)->hasStatus(?P,Falling)*

Sitting *(Rule 3.12-d) WSN(?W), ComposedOf(?C, ?S), usedFor(?S, LinearAcceleration), hasData(LinearAcceleration, a), hasData(LinearAcceleration, ax), hasData(LinearAcceleration, ay), hasData(LinearAcceleration, az),hasMesurement(a, ?mesa), hasMesurement(ay, ?mesay), hasMesurement(az, ?mesaz), hasMesurement(az, ?mesaz) ,hasOwner(?W, ?P), isFormedof(?W, ?C), uses(?P, ?C), Engagedin(?W, LinearAcceleration), TimeStamp(?mesa, ?t1), TimeStamp(?mesax, ?t1), value(?mesa, ?vala), abs(?vala,?vala), value(?mesax, ?valax), abs(?valax,?valax), value(?mesay, ?valay), abs(?valay,?valay), value(?mesaz, ?valaz), abs(?valaz,?valaz), greaterThan(?vala, 9), lessThan(?vala,10.5), greaterThan(?valax,6), lessThan(?valax, 8), greaterThan(?valay,2), lessThan(?valay, 5), greaterThan(?valaz, 2), lessThan(?valaz,5) -> hasStatus(?P, Sitting)*

Moving *(Rule 3.12-e) WSN(?W), ComposedOf(?C, ?S), usedFor(?S, LinearAcceleration), hasData(LinearAcceleration, a), hasOwner(?W, ?P), isFormedof(?W, ?C), Engagedin(?W, LinearAcceleration), uses(?P, ?C), TimeStamp(?mesa, ?t1), value(?mesa, ?vala), abs(?vala, ?vala), greaterThan(?vala, 10.5), lessThan(?vala, 9) -> hasStatus(?P, Moving)*

DangerHR *(Rule 3.12-f) Patient(?P), ComposedOf(?C, ?S), measures(?S, ?mes), usedFor(?S, HR), uses(?P, ?C), value(?mes, ?val), Age(?P, ?Age), multiply(?x, 0.7, ?Age), subtract(?y, 208, ?x), greaterThan(?val, ?y) -> hasStatus(?P, DangerHR).*

A composite process fall detection is created. This process is composed of two consecutive simple processes: "AccSmallerThreshold" and "AccGreaterThreshod". The minimum and the maximum thresholds are respectively the input parameters for the "AccSmallerThreshold" and "AccGreaterThreshold". These parameters are not given statically in order to permit the service provider to set them based on his/her requirement. The output parameter for these simple processes is a Boolean value indicating if the captured acceleration is smaller (or greater) than the threshold.

The "AccelerationBaseFallDetection" composite process has the patient ID for input parameter and is expanded to the "AccSmallerThreshold" and the "AccGreaterThreshold". It has a call notification as effect.

At this level, all the services should be discovered. The service grounding should now be described for the services. For pre-validation only, we considered that the end users can access the ontology information using the CoAP protocol. More details about the service grounding are given in Chapter 4 while presenting the overall architecture. A new individual "CoAPCommunication" of type CoAP is created having 5683 as port number. Moreover, the corresponding object property *supports* is created between "AccelerationBaseFallDetection" and "CoAPCommunication".

3.5.3 Validating the Smart Home ontology using Pellet and SPARQL

Fortunately our proposed ontology is well classified. Figure 3.19 depicts that the status of the patient is inferred. As the values of a_x and a_z are zeros, while a_y is 10, the patient is standing. His heart rate is 200, greater than the maximum heart rate. Thus "hasStatus DangerHR" is logically inferred. For node discovery, the following SPARQL query is executed.

(SPARQL 3.1) SELECT ?Patient ?Nodes WHERE { ?Patient wsn:uses ?Cluster. ?Nodes full:BelongToCluster ?Cluster}}

The result of the query is shown in Figure 3.21. The first patient is using the localization sensors, the temperature sensor, the heart rate sensor and the accelerometer sensor. The second patient uses only the localization sensors.

Figure 3.19-WSN1_Patient1 inferred properties

Figure 3.20 depicts the information inferred for an atomic service, in particular, the "IndoorLocalization" service.

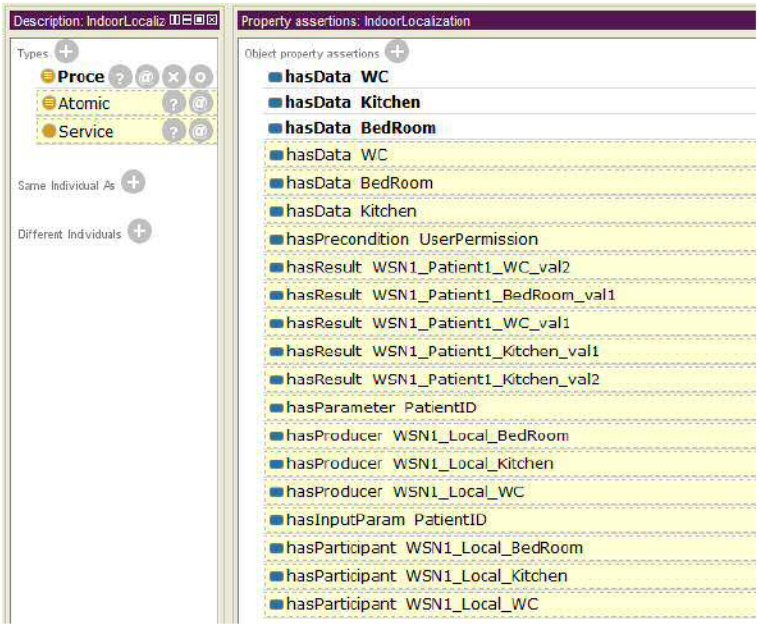

Figure 3.20-Indoor Localization inferred information

Patient	Nodes
WSN1_Patient1	WSN1_Local_Kitchen
WSN1_Patient1	WSN1_Local_BedRoom
WSN1_Patient1	WSN1_Local_WC
WSN1_Patient1	WBAN1_Patient1_Temp
WSN1_Patient1	WBAN1_Patient1_Acc
WSN1_Patient1	WBAN1_Patient1_HR
WSN1_Patient2	WSN1_Local_Kitchen
WSN1_Patient2	WSN1_Local_BedRoom
WSN1_Patient2	WSN1_Local_WC

Figure 3.21-Node Discovery using SPARQL

To discover the location of the patients (in particular Patient 1), the following SPARQL query is executed. The result shown is Figure 3.22 ensures that the location of the patient is correct due to the use of the object property *measuredFor*.

(SPARQL 3.2) SELECT DISTINCT ?Patient ?Location ?Time WHERE { ?Patient wsn:uses ?Cluster.?Nodes full:BelongToCluster ?Cluster.?Nodes full:usedFor smart:IndoorLocalization. smart:IndoorLocalization process:hasData ?Location.?Location process:hasMesurement ?Mes.?Mes full:measuredFor ?Patient.?Mes rdf:type process:Measurement; process:value true; process:TimeStamp ?Time . Filter (?Patient = smart:WSN1_Patient1)}

Patient	Location	Time
WSN1_Patient1	Kitchen	"2015-04-28T04:00:00-05:00"

Figure 3.22-Patient Indoor Localization SPARQL query

Unfortunately, the status of the patient cannot be deduces using Protégé because the SPARQL engine in Protégé doesn't retrieve inferred data.

To discover the services enabled by the patient, the following SPARQL is executed:

The result depicted in Figure 3.23 shows the services enabled by WSN1_Patient1.

(SPARQL 3.3) SELECT ?Patient ?Service WHERE {?WSN wsn:hasOwner ?Patient.?WSN smart:Engagedin ?Service.?Patient smart:enable ?Service.}

Patient	Service
WSN1_Patient1	HR
WSN1_Patient1	AccelerationBaseFallDetection
WSN1_Patient1	LinearAcceleration
WSN1_Patient2	IndoorLocalization

Figure 3.23-Service Discovery SPARQL query

Note that we limited our test on few SPARQL queries that don not need inferred knowledge because, as mentioned previously, SPARQL engine in Protégé is not capable of retrieving inferred data. A fully detailed scenario will be given in Chapter 4 where the SPARQL queries are used in Jena API which is able to retrieve inferred data.

3.6 Summary

In this Chapter, new upper service ontology "MyOntoService" is proposed, specified and formalized. It is divided into three main sub-ontologies: "MyOntoServiceProfile", "MyOntoServiceProcess", and "MyOntoServiceGrouding". Reproducing the same concept as OWL-S ontology for automatic service discovery, and relying on MyOntoSens ontology, this new upper service ontology enabled the automatic discovery of nodes and sensors. The ontology is made up of 1365 axioms, 121 classes, 120 object properties and 119 data properties. This service ontology provides a semantic level for WSN related services classification and interconnection. In that way, any external user/application/software can discover the service, its QoS and QoI, who are the participants (nodes, patients, etc.), and how it can be invoked (input/output parameters, post-condition and effect). Moreover, it indicates which protocol can be used to access the service (CoAP, HTTP, SOAP, etc.). General SWRL rules related to WSN were added like the rules for the atomic services. More specific rules related to "Smart home monitoring" systems are introduced. Finally, to pre-validate the ontology, the individuals related to two patients monitored in a smart home are considered. One patient was equipped with a heart rate sensor and a Smartphone containing an acceleration sensor and a temperature sensor. The other patient needs only to use the indoor localization service. For that purpose, three Bluetooth Beacons are used for Indoor Localization. The system inferred the status (sitting, lying, falling, in danger heart rate zone) of the user based on SWRL rules. The services as well as the nodes were discovered using SPARQL queries.

The technical report of oneM2M which studies the abstraction and semantic enablement in M2M networks defines key issues on standardization of semantics and ontology (65). Our proposed ontology directed these points as follow:

1- It enables the sharing of common understanding of information among M2M nodes due to the use of semantic profile description for the services offered by these nodes. We think that if several different Home Automation Systems are built relying on MyOntoService Ontology, they could share their data, extract and aggregate information using SPARQL query in order to discover the services and its description.

2- It allows the integration of existing ontologies like openEHR ontologies, location ontologies, and person ontologies as mentioned previously.

3- It is flexible enough to integrate new ontologies and technologies. For. Example, if new protocols of communications appear in the market, those new protocols can be described in the interface class (MyOntoSensNode ontology). Likewise, if a new message protocol is proposed, it can be defined in the service grounding class (MyOntoServiceGrouding ontology).

4- It permits the analysis of domain knowledge, provides the possibility to import domain ontology, while describing a service profile, and to use domain specific SWRL rules.

5- It adds a semantic annotation for all the devices and components characteristics of M2M networks. The process of annotation is detailed in Chapter 4.

6- It simplifies the generation of new resources due to its modularity and the use of SWRL rule that inferred all required relations between different resources.

7- The reasoning mechanism, as well as the enablement of semantic rules, is represented in Chapter 4.

In summary, having a complete, well-structured and consistent ontology is not sufficient to provide a complete solution for automatic node discovery, sensor discovery, data aggregation and network management. Therefore, a complete general architecture is designed and detailed in Chapter 4. This architecture clarifies the process of semantic annotation, semantic retrieval, data collection, and node discovery, service discovery, reasoning and adding the missing semantic rules.

CHAPTER 4- GLOBAL ARCHITECTURE

4.1 Motivation & Needs

WSN services are presented as close solutions where the users must adopt a low-level sensor network configuration, and use a specific application coupled with this implemented network configuration. This application prohibits the possibility to benefit from collected data among different solutions and service providers. For this reason, a general architecture should be designed to achieve interoperability, not only between WSN's components, but also between different solutions and applications. This architecture should emphasize the cooperation between different WSNs and should also lead to new cross domain applications in order to integrate the WSN's in the Web of Things (WoT). In addition, this architecture should:

- Enhance the nodes registration process toward zero configuration and automatic nodes discovery,
- Handle urgent cases and notifications,
- Provide dynamic reconfiguration of the network when necessary to extend the lifetime of the network,
- Provide faults' management,
- Address the security, privacy, confidentiality,
- Ensure the freshness of the data and services,
- And especially enable the automatic service discovery and description.

To achieve these goals, SOA approaches should be envisioned. A semantic representation of the information is also the key issue to manage the interoperability between different vendors and applications. Furthermore, to provide a smart and context aware environment where computers will accomplish tasks instead of people, the multi-agent architecture is the most promising technique. This architecture is dynamic, where agents are invoked when needed and can communicate to invoke other agents. These agents will work autonomously and cooperatively. In addition, the multi-agent architecture provides a good level of granularity and redundancy where agents can be implemented on different devices; thus enabling a distributed computing architecture.

This Chapter presents an overall architecture that relies on the previously proposed ontologies: MyOntoSens and MyOntoService. It uses the SOA technique, in addition to the multi-agent architecture. It can be considered as the first step toward a standard architecture to integrate sensors/actuators and data into the virtual world of the WoT. It defines the structure of the overall systems and identifies the relevant functions, information flows and interfaces. Firstly, an investigation about the existing architecture for WSN and medical systems is depicted. Secondly, our proposed general architecture is defined and detailed showing the overall mechanism and process for automatic node/service discovery as well as WSN's management. Finally, a specific use case already envisioned in Chapter 3, the smart home monitoring for elderly, will be carried out for testing and qualifying our architecture in a real environment.

4.2 State of art

While some researches were focused on integrating the principle of Service Oriented Architecture (SOA) for WSN, others have proposed to add semantic meaning to the description of the nodes and data generated by WSN's components. These efforts are summarized in this section. Moreover, being interested in biomedical systems as application domain, a review of medical systems architecture is presented.

4.2.1 SOA for WSN

Lan *et al.* (66)propose a service oriented architecture for wireless sensor networks as shown in Figure 4.1.

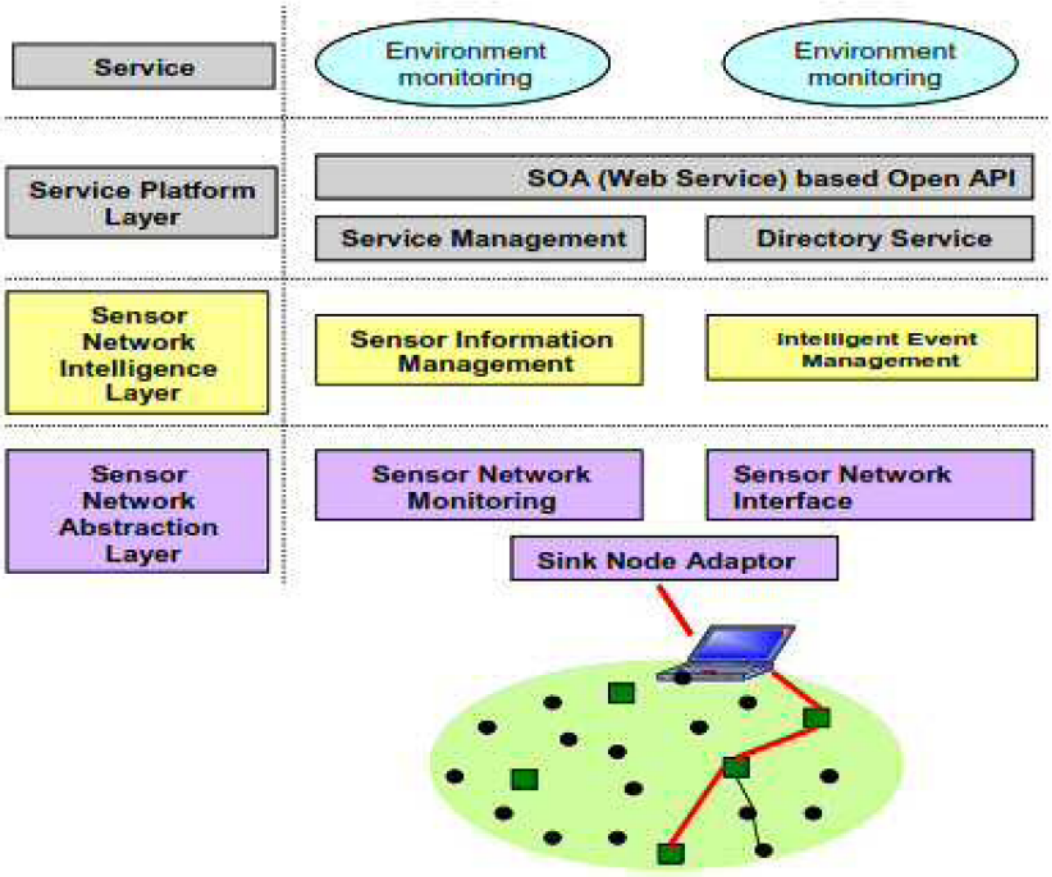

Figure 4.1-Service Oriented architecture for WSN (66)

The architecture is split into three main layers:

- The Service Platform Layer which provides an application interface for various services such as the discovery of components or sensor network.
- Sensor Network Intelligence Layer which provides intelligent sensing data processing, intelligent event processing, and context information processing.

- Sensor Network Abstraction Layer which provides wireless information infrastructure abstraction and sensor network monitoring.

Avilés-López *et al.* (67) propose TinySOA, a service oriented architecture for WSNs as shown in Figure 4.2.

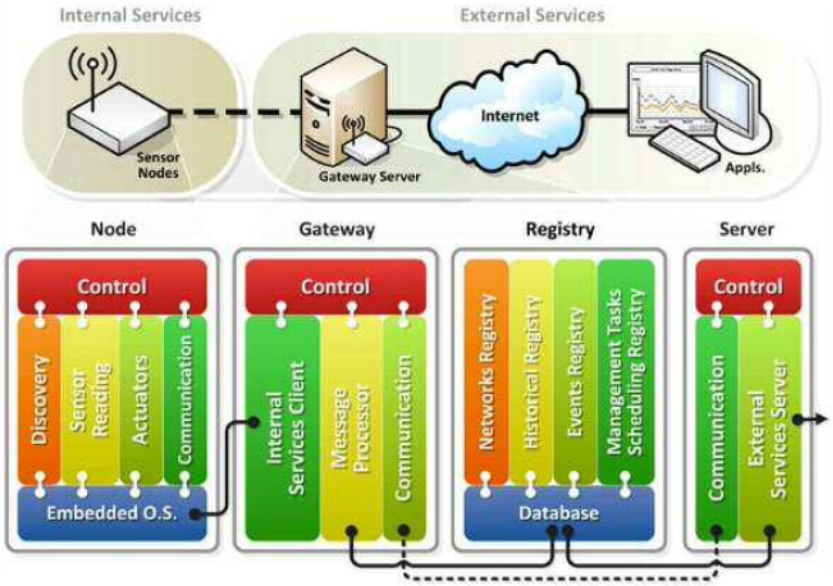

Figure 4.2-TinySOA architecture (67)

The Node encompasses all the functionalities of sensing nodes and networks. The gateway, at start-up, receives the registration messages with the single capabilities of each node. This gateway then aggregates the received node capabilities, and deduces what services the full sensor network can provide. All information about the infrastructure is stored in the registry. The server acts as a provider of web services, abstracting each available WSN as a separate web service. The registry provides an interface to consult the services offered by each of the networks checks the registry, consults and registers events and maintenance tasks. Additionally, an information web service, in which clients can know how many different network web services are available and how to access them, is set up.

TinyVisor was implemented for testing. At start up TinyVisor asks for the URL (Unique Resource Locator) of a TinySOA server. This server URL must be provided in order to locate the information and network web services. Once connected to the server, TinyVisor proceeds to open the information web service and sends a request to know how many different network web services are currently available. An interactive dialog is then displayed. This dialog shows the information related to the available WSNs including the name, description, and web service URL. It is then possible to select the network that is going to be used for monitoring and visualization. Once this network is selected, the information regarding to its nodes and the sensed data can be visualized, either in data mode graph mode, or topology mode. Figure 4.3 depicts the functions that can be called from any program via web services interface to access WSNs.

Figure 4.3-TinySOA functions that can be called from any program via web services interface to access WSNs (67)

Delicato *et al.* (68) propose another service oriented architecture for WSN as shown in Figure 4.4.

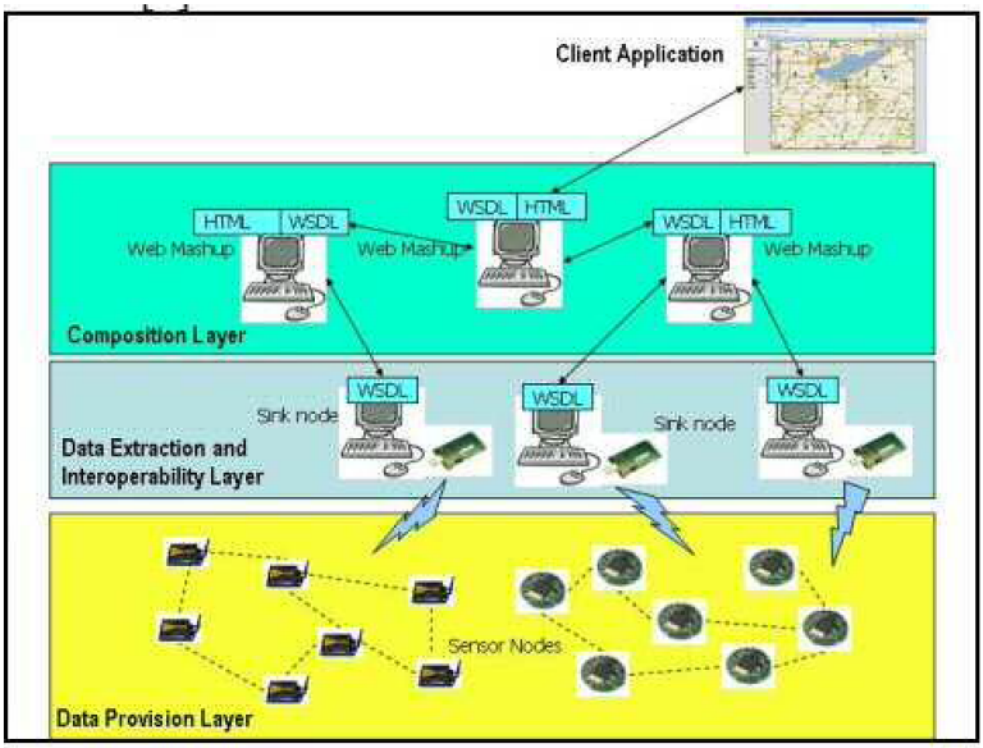

Figure 4.4-Service Oriented Architecture for WSN (68)

A sensor node can play the role of sensing device, router, and/or data aggregator. The Sink node provides the collected information to other architectural components. Furthermore, it can aggregate data. The Composition Layer consists of Web Mashups. This layer provides value-added services through the composition of data extracted from different WSNs using the sink node common interface. Such Mashups allow different levels of both visualization and processing of a WSN. Web Mashups are published and discovered using the Mashup Catalogue

that plays the role of service registry. The Mashup Catalogue can be implemented as a regular UDDI (Universal Description, Discovery, and Integration) register. The communication between Sink nodes and the sensor network is accomplished using a WSN specific data dissemination protocol and formatted as XML messages. HTTP binding is used to encapsulate the message exchange between Sink and sensor nodes. Figure 4.5 depicts the communication stack between sensor, sink and Mashups.

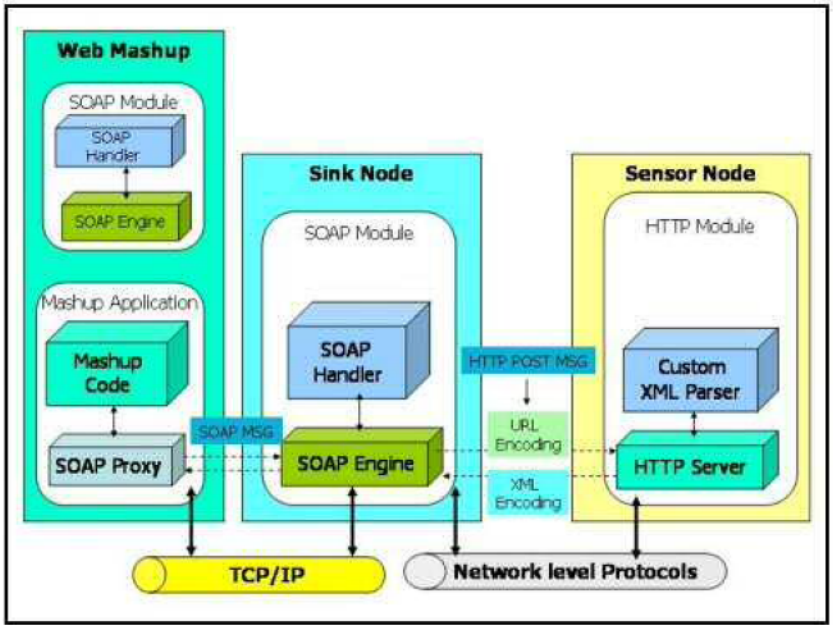

Figure 4.5-Communication Stack (68)

Ghobakhlou *et al.* (69) have based their architecture on the XML-based of the OGC SWE protocols and encodings. To overcome the overhead caused by XML exchange, they proposed to interlink OGC SWE semantics for sensor descriptions and observations (OGC Sensor Observation Service, SOS), as well as sensor configuration and planning (OGC Sensor Planning Service, SPS), with the open Message Queue Telemetry Transport (MQTT) protocol. A mapping of the complex and generic SWE semantics within MQTT implementation was developed.

In 2009, K. Khedo *et al.* (70) proposed MiSense, a service oriented component based middleware architecture for WSN aiming to provide an abstraction layer between applications and network layer in WSN. MiSense relies on the publish/subscribe service mechanism. It involves four layers:

- The communication layer responsible of the subscription and notification events,
- The resource management layer that manages the access control to the required resources in order to execute a process required by an application,
- The common service layer which provides the common services for a WSN (data aggregation, topology management and routing),
- And the Domain Layer where specific services are defined like, e.g., data fusion.

In 2014, Singh *et al.* (71)suggested a flexible service oriented network architecture for WSN which supports service discovery, service abstraction, QoS and transparency, as well as interoperability among services. The main players in the architecture are:

- Service Providers: provide the service to application/users. The service providers can develop new service based on predefine, precompiled and pre-composed methods as per the requirement parameters of users.
- Service Brokers: match the required service to a pre-defined existing services using protocol graph generator. It translates the messages to the adequate format and transmits it to the requestor.
- Service Users: request the service. It can be a parametric user which knows the technical details of the architecture, or a non-parametric user ignoring the details and requesting common services like monitoring.

4.2.2 Semantic SOA for WSN

Amato *et al.* (72) proposed an innovative architecture for risk management that relies on services. The proposed architecture permits the collection and the aggregation of raw data in order to create end-users application using web services. Figure 4.6 depicts the general view of the proposed architecture.

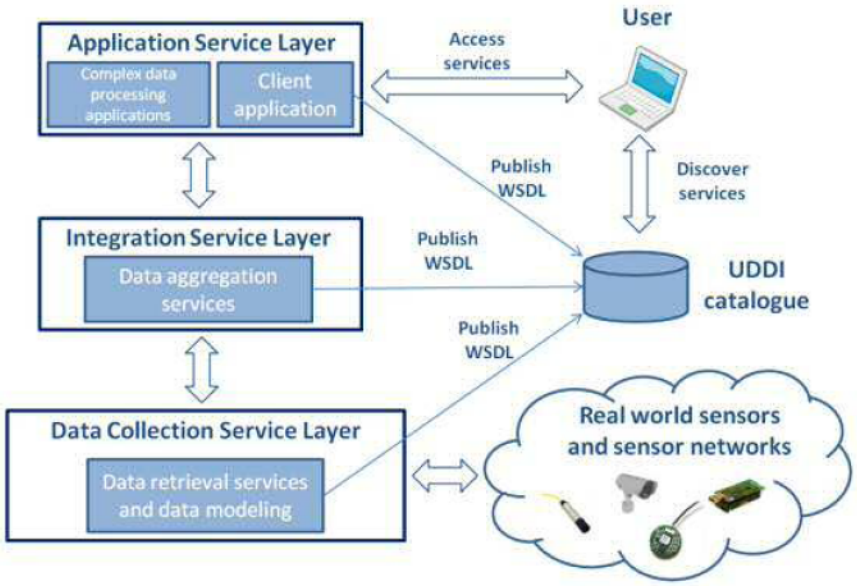

Figure 4.6-Semantic SOA Overview Architecture (72)

The data collection layer interacts directly with the WSN, collects the raw data, and enriches these data with ontological modelling techniques. This layer offers the service of converting data into XML/RDF data model.

The Integration Service Layer aggregates data and clusterizes the data according to a defined data model, then delivers it to the application service layer.

The application service layer invokes the service, in general, via the WSDL standard.

Each service has its own security policy and it is published in a public registry. To integrate data from heterogeneous source the wrapper-mediator paradigm has been used. Each wrapper explores and monitors the local sensor network and sends to the mediator an appropriate description of the related information according to a common data model. Figure 4.7 summarizes the wrapper-mediator functionality within the Collection and Integration services.

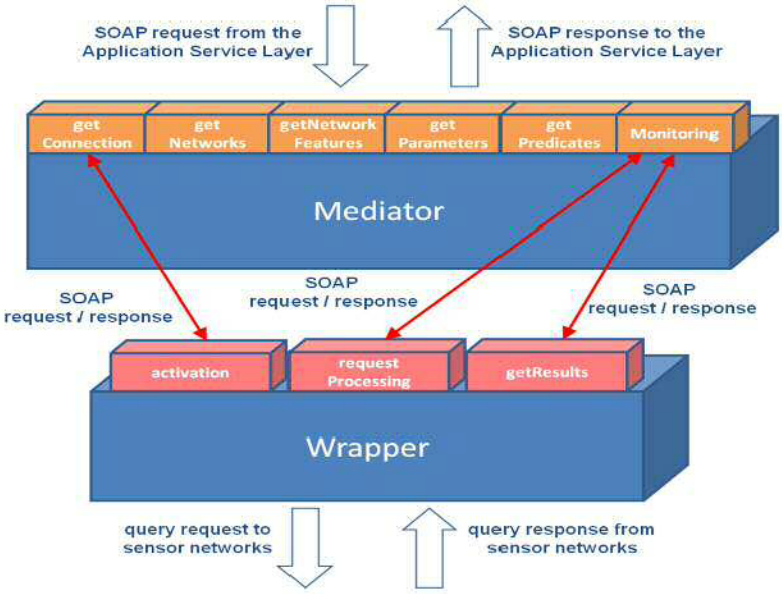

Figure 4.7-Collection and Integration Services (72)

For the discovery and registration of a specific wrapper, the "*getConnection*" function is used. It then activates the service provided by the wrapper module. The "*getNetworks*" gives as output the list of all networks connected to the system, their topology, and basic information such as network identifiers, number of sensors and maximum depth for each network. The "*getNetworkFeatures*" takes as input parameter the network identifier and returns the system's type of system, middleware, number of sensors, maximum depth, IP address, registration time and communication ports, as well as information about the base station (frequency, communication link). The "*getParameters*" returns the parameters of the sensor (identified by the network identifier, cluster and sensor identifier), like the free memory, the voltage and the channel quality. The "*getPredicates*" gives all the physical variable a sensor (identified by the sensor identifier) is able to measure. Finally, the monitoring task queries a sensor or a network, invokes the "*requestProcessing*" and "*getResults*" services to get the results related to a specific query.

In conclusion, to add semantic meanings to the sensed data, the sensors add-on (data aggregator) retrieves samples from sensor systems and converts them into an XML format, according to defined translation rules. The XML-RDF Wrapper translates the file into an RDF document, according to a set of sensor domain ontologies. Finally, the O&M Data modeller retrieves an RDF document and converts it in an O&M standard XML document.

OSWA is an extension of SWE proposed by NICTA (National ICT Australia Ltd) University of Melbourne (73). Its main aim is to combine SOA with SWE. OSWA defines four different layers as shown in Figure 4.8. In OSWA, the service layer reuses the services described in SWE

and specifies how these services can be implemented in real scenarios. It uses WSDL (Web Services Description Language) and UDDI for service discovery. SCS provides interface to both streaming data and query based sensor applications that are built on top of TinyOS and TinyDB respectively. Only two features of SPS were implemented: "*getFeasibility*" and "*submitRequest*". The WNS offered two features: "*registerUser*" and "*doNotification*". For user registration a JDBC-based "UserAccountManager" has been implemented. Moreover, the email protocol is used to notify clients. Sensor Repository Service is deployed as a web service that can be accessed via SOAP messages.

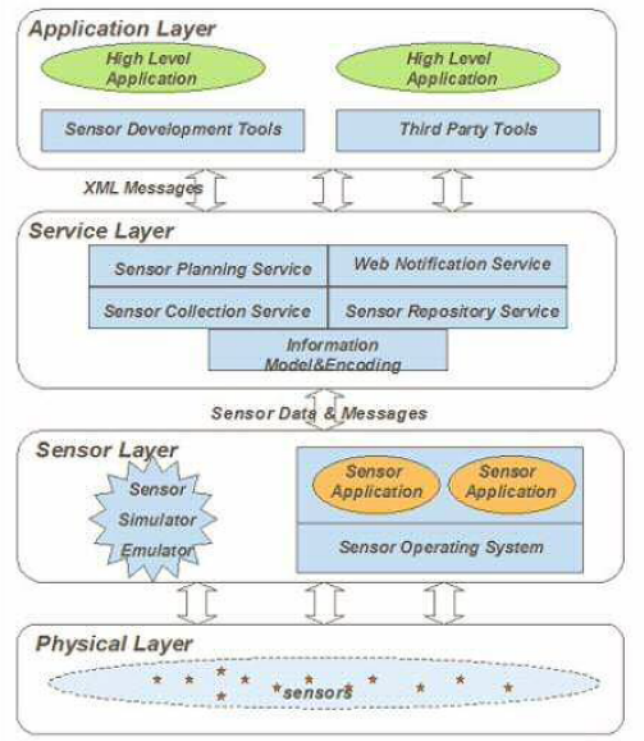

Figure 4.8-OSWA layers (73)

4.2.3 SOA for WBAN and medical systems

4.2.3.1 ContoExam (74)

A general architecture for examinations (ContoExam ontology) has been proposed in (77). It relates examinations to controlled domain vocabularies in e-health (such as LONIC[14] and SNOMED-CT[15]), adds context information about the examinations, and relates domain properties, such as used in epilepsy, to other system of properties such as the Information Standards Quarterly (ISQ). Figure 4.9 depicts the classes' hierarchy of the ContoExam ontology. Figure 4.10 shows how semantic interoperability can be split into 5 layers. The lowest layer, the repository schemata, represents semantics concerning the technological details about data accessing: how to read sensor data; syntactical details of how data are addressed; specifics data

[14] https://loinc.org/discussion-documents/document-ontology/loinc-document-ontology-axis-values/subject-matter-domain

[15] http://www.nlm.nih.gov/research/umls/Snomed/snomed_main.html

about how to store a patient observation according to e-health standards assessments. The syntactic interoperability is assured by driving the management, monitoring and control of all the data handling of the previous layer. The domain-variant ontology layer represents the semantics for communication outside the local system by representing the domain's specific properties. The domain-invariant ontology layer enables the sharing of those concepts. For example, for examination concept, ContoExam ontology represents the domain abstract ontology.

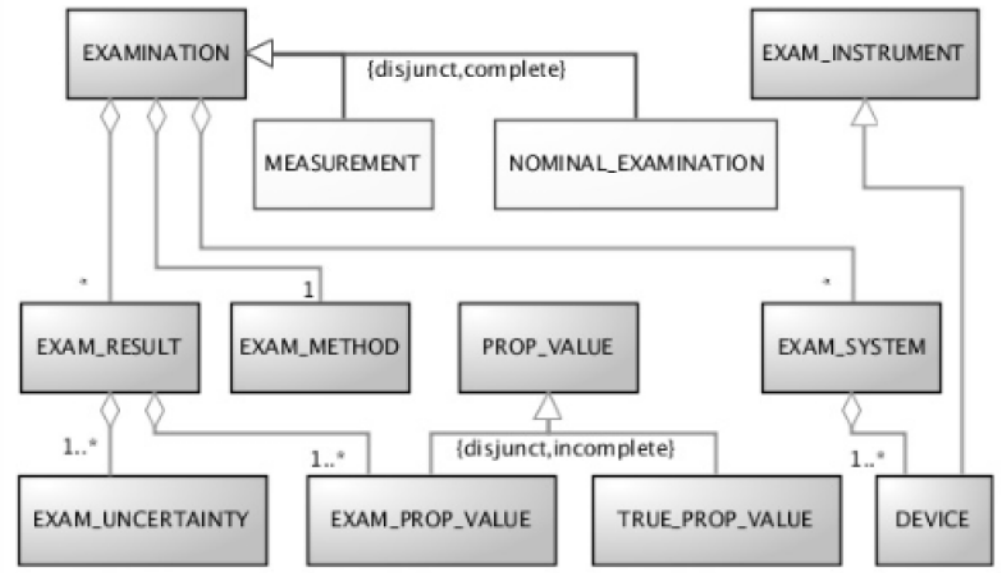

Figure 4.9-ContoExam classes (74)

4.2.3.2 Continua Health Alliance

The Continua Health Alliance is a non-profit industry organization containing about 200 members across the world aiming to 'develop and publish design guidelines that combine and apply existing technology standards to achieve end-to-end, plug-and-play interoperability in personal connected health. Figure 4.11 shows the Continua Alliance global healthcare architecture. The IEEE 11073 standards are based on an object-oriented domain information model (DIM) that contains 8 packages dedicated for medical, patients and device interoperability (44). Its primary goal is to standardize how data move from a sensing device (source) to a receiving device (sink), and what format is used. These standards thus provide interoperable terminology for the data (each medical term is represented by a code formed of 32 bits). For each device type, a device specialization that utilizes the common framework to model the specific data from device type is created. The service model defines how the information model should be accessed by the receiving device based on event reports. Moreover, the communication model defines the way of interaction between the service model and the underlying transport layer. The optimized exchange protocol is designed in an optimized way due to the following factors:

- The configuration parameters are exchanged only at the beginning of the communication,
- The sender/receiver can reuse a previously agreed configuration,
- The sender/receiver can use a predefined standard configuration,
- The sender/receiver can choose between multiple events' reports.

The Bluetooth Special Interest Group (SIG) on June 2008 releases the Health Device Profile (HDP) that allows the consumer to easily connect any two devices that support the medical device profile. The Bluetooth LE defines the Generic Attribute Profile (GATT). The Profiles are high level definitions that define how services can be used to enable an application or a use case.

The Continua End-to-End (E2E) Reference Architecture (76) defines five network interfaces that connect the devices to a Reference topology:

- The Peripheral Area Network Interface (PAN-IF) connects an application hosting device to a PAN device, which is either a sensor or an actuator,
- The Touchable Area Network Interface (TAN-IF) connects between touch area network (TAN) health devices and application hosting devices,
- The Local Area Network Interface (LAN-IF) connects an application hosting device to a LAN device. This device aggregates and the bound PAN devices' information. A LAN device can also implement sensor and actuator functionality directly,
- The Wide Area Network Interface (WAN-IF) connects an application hosting device to one or more WAN devices. A typical WAN device implements a managed-network-based service,
- The Electronic/Personal Health Records Network Interface (xHRN-IF) enables patient-centric data communications between a WAN device and a health-record device, typically at the boundary of the personal tele-health ecosystem.

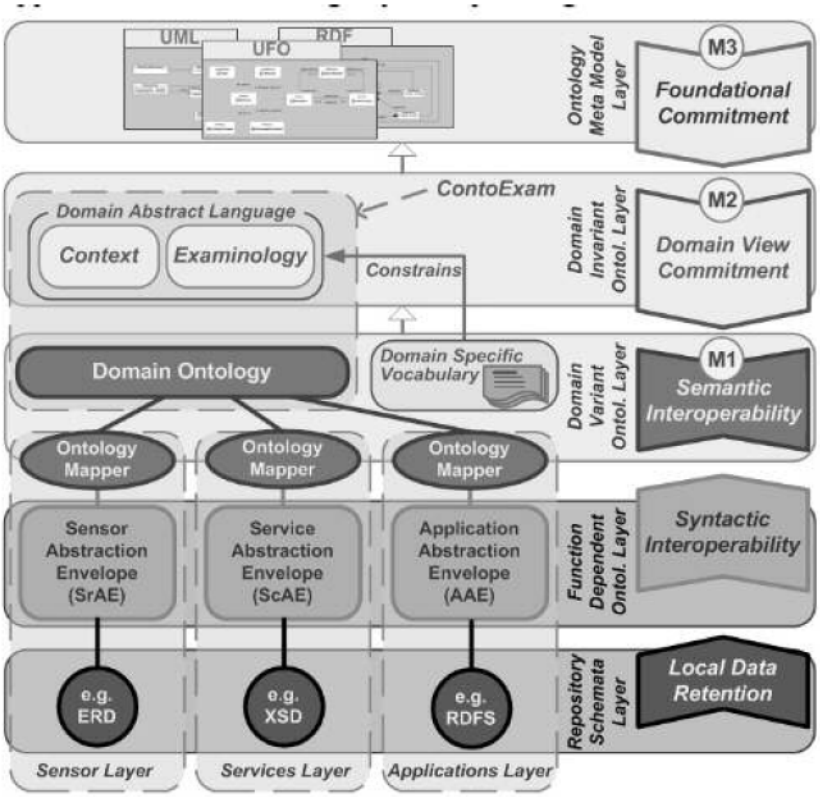

Figure 4.10-Architectural design of semantic interoperability (74)

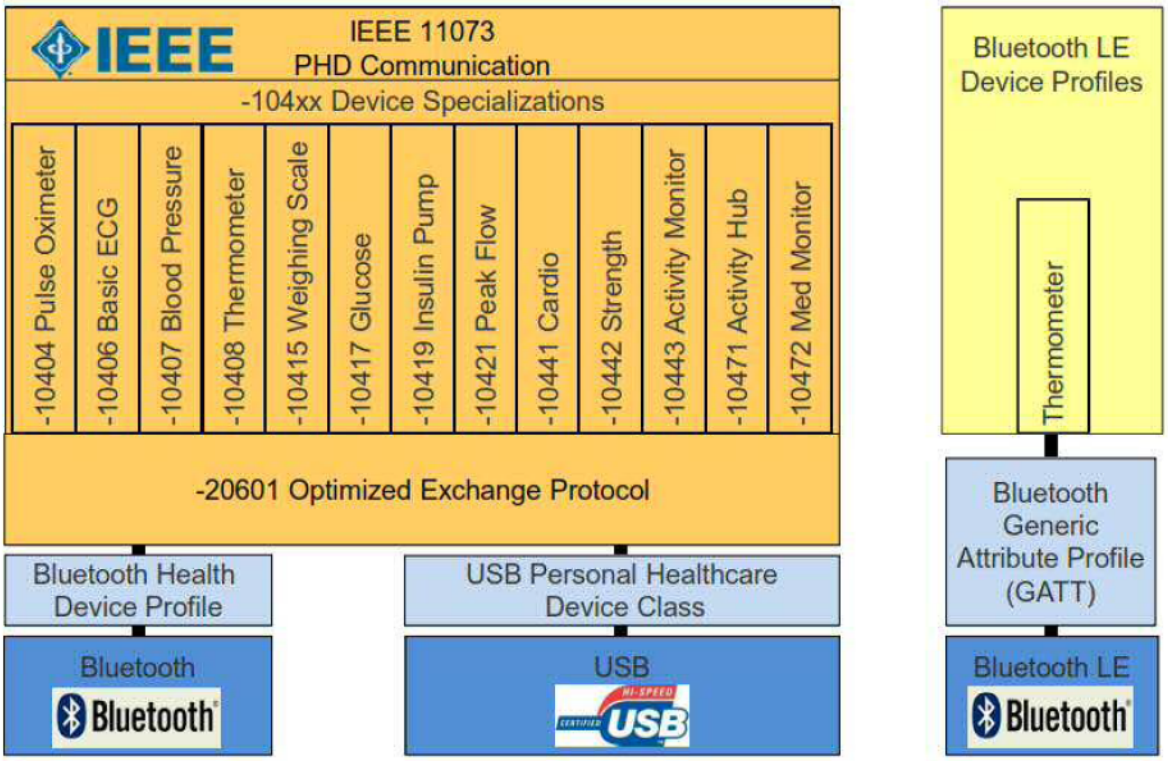

Figure 4.11-The Continua Alliance global healthcare architecture (75)

4.2.3.3 MedCom (77)

The Danish healthcare data network MedCom was developed in 1994. Starting with its first project, MedCom1 focused on developing communication standards for the interaction between medical practices, hospitals and pharmacies, based on Electronic Document Interchanged (EDI). The implementation and consolidation of the project began with Medcom2 in 1997-1999. In 2000-2001, MedCom3 focused on the quality of services and diffusion. With the evolution of the Internet, MedCom 4 adopted Internet and web based technologies in 2002-2005. MedCom 4 periods involved the development of XML standards for EHR data exchanges. In cooperation with the Ministry of Science Technology and Innovation (MVTU) and suppliers of IT systems to the healthcare sector (among others), MedCom has drawn up a proposal (MedCom 5) in 2006-2007 for the "Good Web Service" for general application in the healthcare sector based on the developed XML standards. The proposal is based on a service oriented IT architecture, and the recommendation is for this architecture to be common to the public sector. All clients' requests are converted to XML SOAP messages and sent to a SOAP server for treatments.

4.2.4 Summary

Many attempts have been conducted to introduce service oriented architecture for WSN which consider the sensors/nodes as service providers, the users/applications as service requestor, in addition to a service broker to match the available services with the requested one. Almost all these investigations have been relying on non-semantic matching protocol and only provide limited number of services. Other proposals have integrated the semantic approach, with the combination of service oriented architecture. Those proposals rely on SOAP and UDDI

solutions. OSWA has integrated the idea of ontologies, but it reuses the SSN ontology which does not describe the entire properties of WSN. Some medical architecture has also been proposed but it did not take into consideration the quality of information, as well as the nodes' characteristics. In summary, it does not exist yet a complete architecture that combines the principle of semantic service oriented architecture with the possibility to adapt behaviour of the WSN and the service provider to the frequent change in WSN. This confirms the need to create an overall architecture that:

- Provides the sufficient semantic level for interoperability between different WSNs, and also within the same WSN,
- Enables the idea of plug and play, where the users has just to power on the senor, the service discovery and the selection of the best service being automatically provided to the user regardless of the underneath layers.

This is the main aim of our proposed architecture where we re-used the idea of ontology wrappers defined in ContoExam (74), the service brokers defined in the flexible service oriented network architecture (71), and the layering defined in OSWA (73) . The details of this architecture are given in the next sections.

4.3 The Layering Architecture

To improve maintainability and flexibility, the proposed architecture is designed as multiple layers communicating with each other. Each layer can be implemented independently in a device. Thus, it allows for constrained devices to include just the required functionality. The architecture consists of four layers: the data provisioning layer; the semantic layer, the service layer and the application layers. Figure 4.12 depicts these four layers.

4.3.1 Agents Definitions

Let's start by defining the components of this architecture. In fact, this architecture is built of various software agents in order to provide granularity, flexibility and the ability to adapt to pervasive environment such in WSNs.

An software agent is a piece of software that functions as an agent for a user or another program, and working autonomy and continuously in a particular environment. They are invoked by a task, reside in wait status, do not require user interaction and can have a starting condition. In our architecture, an agent can invoke another agent and one agent can be used for many functionalities.

Scanners Agents:

The scanners are responsible for the required data retrieval from the physical nodes/users or applications/software. They are also responsible for passing this data out to the semantic wrappers. In fact two types of scanners exist: Data Scanners and Service Scanners.

Data Scanners are in listening mode waiting to receive data from the Data Provision Layer. For each communication protocol a data scanner agent should be implemented. The data scanners link the Data Provision Layer to the Semantic Layer. For example, if a Smartphone is receiving data from Bluetooth LE device and at the same time, if it is communicating with a server via HTTP, two scanner agents should be set on the mobile: the Bluetooth LE Data Scanner and the

HTTP Data Scanner. Because the widely used communication protocols in WSN are ZigBee, Bluetooth LE and WIFI/3G/4G, our architectures encloses now a ZigBee data scanner, a Bluetooth LE Data Scanner and, for the wide area wireless communication, HTTP Data Scanner.

Service Scanners are in listening mode waiting to receive data from application users. They play the role of the bridge between the application layer and the service layer. Regarding the service scanners, the CoAP Service Scanner and the JSON Service Scanner agents have been developed in our architecture because we prefer to use the Restful architecture to provide the WSN services to external users. This choice is motivated by the lightness, easy implementation, flexibility of Restful architectures, and their loosely coupled client/server communication mode (30). This is essential in heterogeneous network dealing with users who ignore the underneath techniques used in WSN (like the non-parametric users cited in MiSense Architecture).

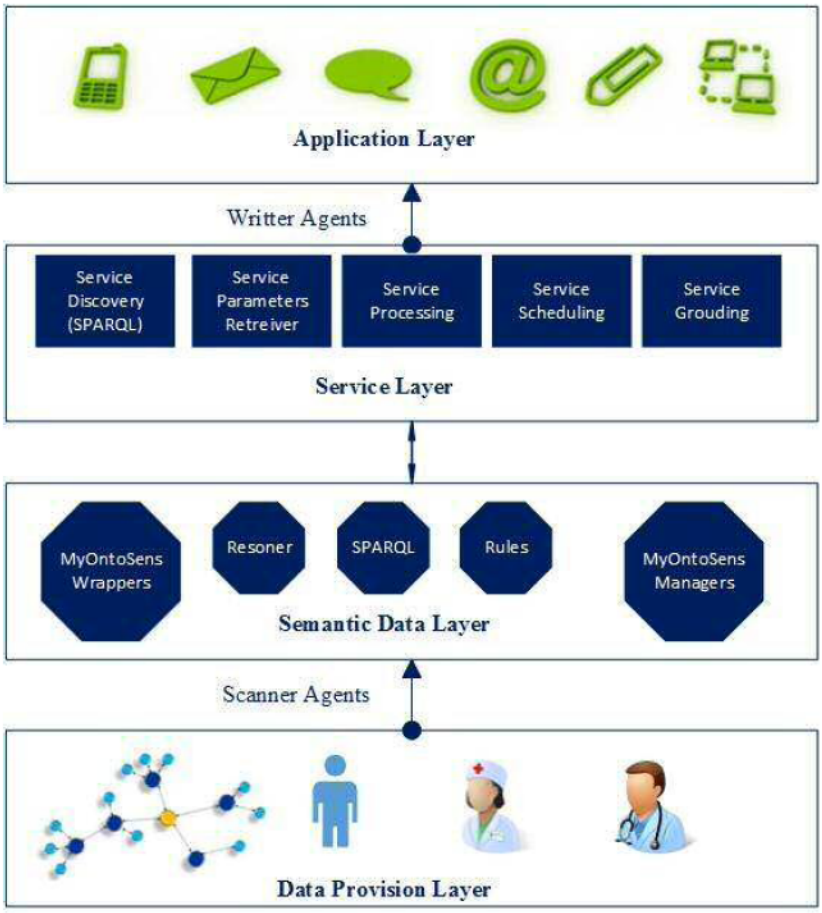

Figure 4.12-OntoSensArchi Layers

In summary, the Scanner agent decapsulates the message sent form WSN's nodes or outside users and invokes the adequate Semantic wrappers in case of Data Scanner or the suitable Service Agent in case of Service Scanner.

Writer Agents:

The Writer agents are used to provide data from the external users or to manage the WSNs. Their main functionality is to encapsulate messages, respecting the communication protocol used. As

for Scanners, each communication protocol has its own Writer Agent. But because the writers don't have to invoke other agents, we didn't separate the service writer and the data writer.

Semantic wrapper (SW) Agents:

Give semantic meaning to the information based on MyOntoSens and MyOntoService ontologies. It can be seen as a function that maps the raw data extracted from nodes/users/applications (via the scanners) to set of well-formed form of the proposed ontologies. It reads data from the scanner agent and adds the semantic information in the Semantic Layer. These agents are always invoked by the Scanners.

SPARQL Agent:

The SPARQL agent should execute SPARQL query on the overall ontology. It is invoked each time a data should be extracted from the ontology (e.g. node discovery, service discovery, retrieval of measurements, retrieval of the characteristics of a node, etc.)

Authentication Agent:

Its role is to authenticate the users and to verify the permission and accessibility of the users. Many techniques can be used: data base technique, access rules technique and external cloud solutions. The developer can use the information stored in the ontology like username, email, and password in the Person Class to authenticate the user. In addition, it can benefit from the relations *isRelative* and *hasContact* predefined in the ontology to determine the user's permission. In these cases, the Authentication Agent will invoke the SPARQL Agent to retrieve the semantic information from the ontology.

Rule Agent:

The role of this agent is to add new SWRL rules to the ontology. It is very beneficent for domain experts' users like the medical staff to interact with this architecture and add specific domain constraints. Either the users will write the SWRL rule, but this solution is not very professional. Or an SWRL translator is used to transform the constraints set up by the experts to a SWRL rule (e.g. via a Web Interface). The Rule agent should run the Inference Engine to insure that this rule does not cause any inconsistency.

In case the new ontology is well classified, a confirmation message will be sent to the expert. Otherwise the expert will be asked to modify the constraints. This agent is still under development.

Trigger Agent:

The role of the trigger agent is to listen to a certain event and invoke the notification agent or wrapper agents based on the scenario. We are proposing to use the semantic trigger agents that listen on a specific part of the ontology and generate a trigger on change. But, any type of triggering systems can be used without affecting the overall functionality of the system.

Notification Agent:

The role of the notification agent is to send urgent notification or reminder to subscribed users. A MQTT agent, or GCM notification agent, can be used. The service developer can always implement its own notification agent via email, SMS calls, etc.

Security Agent:

It is role is to encrypt/decrypt data based on an external system or on the information stored in the ontology (Security Constraint Class). If the data is secured, this agent is invoked by the scanners to decrypt the messages and the writer to encrypt the messages.

Traffic Management Agent:

The role of this agent is to prioritize the services requested by a user and decide how and to whom the response should be sent. The management is done into two levels:

- Routing management: if we have different routes in the WSN to attempt the users, the traffic management should select the best path. To do so, it will invoke the SPARQL Agent to retrieve information about, for example, the state of the link (*LinkState* Class), the reaming battery of the gateways or the sinks (*MemoEnergy* class), the current mode of a node (*inCurrentMode* object property), etc.
- Service Priority Management: based on the QoS (QoS class), the degree of urgency some services should be executed before others. Thus, the SPARQL Agent is invoked to retrieve the information about the requested services (*hasEffect* Notification should have the highest priority, or from the QoS class). Then, the service priority managers order the execution of the services. It invoked the writers to send responses to the users or execute commands on actuators/sensors.

Service Processing Agent:

This agent is responsible of executing the process. It is preceded by the SPARQL agent who retrieves the service processing description of the requested service. It takes the input parameters, processes data and returns the output parameters. It can be preceded by a pre-condition and can have effect depending on the service process description (MyOntoServiceProcess ontology classes and properties). Each developer conceives its own service processing agent in the language that he/she prefers.

Application Agent:

This agent is left to the programmer and developer in order to display data depending on the end user terminal.

4.3.2 Process and data Flow

The aim of this section is to describe the overall process and interaction of different components/agents defined in the proposed architecture starting from the Setup phase, passing to the sensor enablement phase, service discovery phase, measurement collection phase to finalize with the WSN management phase.

4.3.2.1 Set up phase

The first step in any software application is the Set up phase, where the users provide the adequate information to the proposed architecture. The setup phase can be done using a mobile application or a website or any other solution enabling the users to enter data. If we are dealing with a simple architecture, the main players will be: the hub (in general the Smartphone or the PDA of the users) and the cloud server. Otherwise, in multi-tier architecture, the PDA, the hub

and the cloud servers are engaged in the setup phase. Based on the communication protocol adopted in the WSN, scanners agents and writer agents should be implemented. Let's consider the case of a WSN dedicated for fire monitoring in a forest. A central node or hub (with consistent source of energy) will collect the data via ZigBee protocol, process it, and send it to an external server via WIFI protocol. To do so, ZigBee scanner and writer agents as well as WIFI scanner and writer agents should be deployed in the hub.

First of all, as mentioned previously, the users (patients, medical staff, WSN's responsible party, etc.) will enter the detail of the WSN using a GUI interface. In general, the details are about WSN's owner, WSN's location, WSN's application domain, etc. (information that suits MyOntoSensWSN ontology). If the communication is between the Smartphone and the cloud server using JSON, the JSON scanner agent on the cloud server will receive the data from the PDA, and form the title of the JSON request, identify which semantic wrappers should be called, which in turn, add this information to the semantic cloud server. So the data provision layer and the semantic layer are used in the setup phase. Finally, this phase can be customized by the application's developer based on their need to setup their network. Figure 4.13 depicts the use case of the setup phase.

4.3.2.2 Node Discovery Phase

One of the most complex issues in the integration of WSN in the WoT is the automatic node discovery. The first question is how to detect the presence of a new senor in the network? How to discover the services offered by this sensor? How this sensor is affecting the overall functionality of the system? These questions should be answered with minimum user's interaction. Our proposed architecture responds to these questions as follow. First of all, the Data Scanners agents receive the data from nodes/users. For example, if we are using a ZigBee WSN, the ZigBee Data Scanner agent, which should be implemented in a coordinator, should read the data received by a node, and identify the AT command used in order to call the adequate wrapper agent (10). If a new ZigBee node joined the network, the ZigBee Data Scanner agent will retrieve the node's information (MAC address, channel, Firm version, etc.) and call the Node Wrapper agent. The Node wrapper Agent adds this information to MyOntoSensNode ontology and adds the object properties "BelongToCluster" and "BelongToWSN" to the previously created WSN individual for linking these sensors. In that way, once the SPARQL query dedicated for nodes' discovery is executed, and the nodes will be discovered without the interaction of the user. Same case if we are using Bluetooth LE where the Bluetooth LE data scanner will be implemented on the server. Based on the UUID, this Data Scanner can determine the wrapper agent that should be invoked. More details about the SPARQL query is given in section 4.5.

Moreover, each sensor is used to monitor certain process. Thus, the scanner should identify this information and invoke the Process Semantic Wrapper to assign the process (*usedFor* object property) for each sensor. For example, in Bluetooth LE, the UUID defines if it is used for heart rate monitoring, blood pressure and other. Till this step, the answers to the two first questions are provided: the discovery of the node and the atomic service offered by it. Figure 4.14 depicts the basic activities for node's automatic discovery.

More complicated issue is now how the discovery of this node is affecting the overall network. In general, this question is essential in large sensor network where more than one sensor is used to provide the same data. In this case, the actual status of the sensor (current mode, available

battery lifetime, routing level and others) can play an effective role in the data fusion mechanism, as well as in the management of the WSN. If it is the case (mostly used in ZigBee networks), the MemoEnergy and LinkState wrappers are invoked. And if necessary, the ZigBee writer can be called to put a sensor in sleep mode or change the coordinator in the network due to its limited battery lifetime. The ZigBeeWriter will encapsulate the message in the adequate AT command.

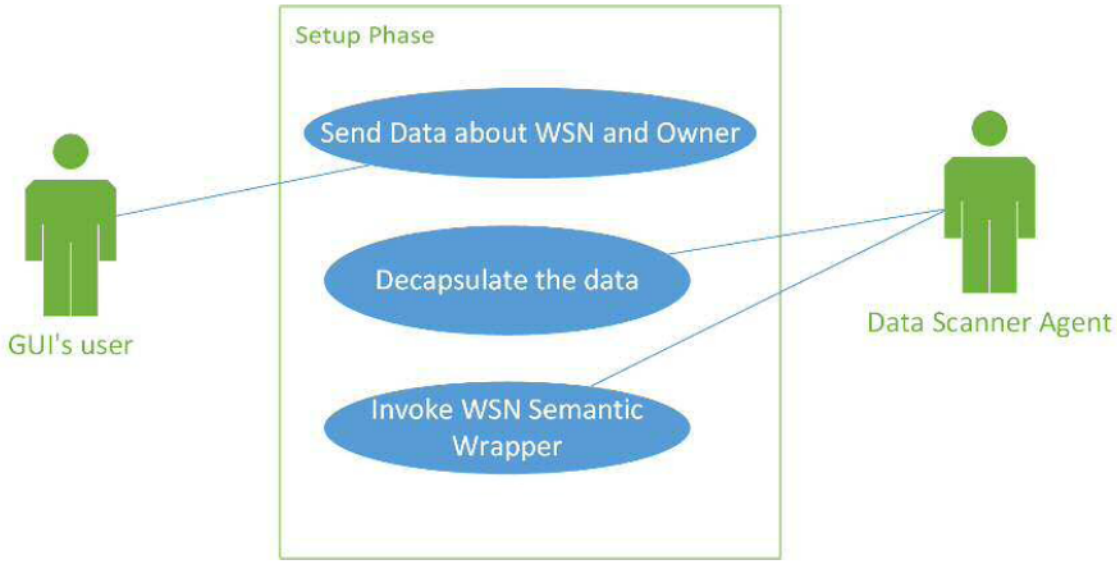

Figure 4.13-Setup Phase Use Case Diagram

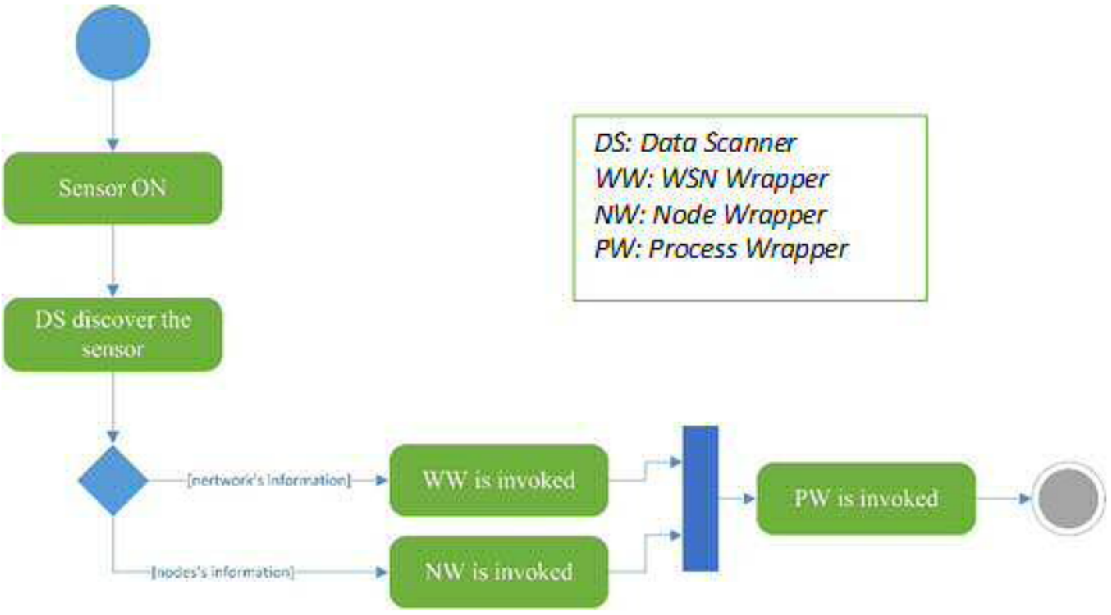

Figure 4.14-Node's discovery basic activities

Fortunately, this step enabled the automatic node discovery encompassing the node's physical characteristics, node's atomic service, nodes' energy characteristics, etc. This information not only permits the discovery of the nodes, but also gives the opportunity to effectively and remotely manage the WSN.

4.3.2.3 Measurement Collection Phase

The measurement collection phase consists of collecting raw sensed values in order to add it to the Measurement class in the MyOntoSensProcess ontology. The collection can be periodic, on event triggering or on request. If it is periodic, the DS (Data Scanner agents) will periodically retrieve the value by invoking the MW (Measurement Wrapper) agent. Figure 4.15 depicts the activity diagram for a periodic data collection. Note that this flow of activities can be done on the server or on the hub depending on the deployed architecture. For example, in a WBAN application, the Smartphone can be the data collector, thus, it will execute this flow of activities.

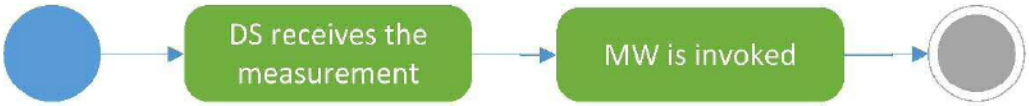

Figure 4.15-Periodic Data Collection

Measurements can be requested directly by an external user (e.g. a doctor request the heart rate of his patient), or when an event occurs (e.g. when a patient falls, we should retrieve his location). We considered that all the semantic processing is on the server, and the hub is the intermediate node between the server and the WSN. In case of a request, the Service Scanner agent will be invoked to retrieve the user's request. Otherwise, the event triggering on the server will be the start point for the measurement collection. A detailed flow of messages is depicted in Figure 4.16. The server will invoke the SPARQL agent to retrieve the measurements. If it exists, the measurements are sent to the application agent to display it (see Figure 4.16, Task 3.a and Mes.2). Else, the measurement should be collected from the sensor. To do so, the server will send the request to the hub (see Figure 4.16, from Task 3.a till Task.4), which in its turn, sends it to the sensor hub (see Figure 4.16, from Mes.4 till Mes.5). When the sensor sends its measurements, the hub forwards it to the server that will add it to the ontology and re-invoke the SPARQL agent hub (see Figure 4.16, from Mes.6 till Task.9).

4.3.2.4 Service Discovery Phase

In fact, once the node discovery phase is done, the atomic services are discovered as previously explain in Chapter 3. Simple services can be discovered base on SWRL rules. For example, if we have a heart rate sensor, we can calculate the calories burned during exercises. Thus a simple calories burned service can be discovered based on a SWRL rule. The architecture is also leaving the developer the choice of how to discover composite and simple services. It can be via a SWRL agent, or a GUI interface where the WSN owner defines new services. In this case, the Data Scanner Agent retrieves the information about the service and invokes the Service Wrapper agent.

Afterwards, the service discovery is done using SPARQL queries. In our architecture, the service registry is the MyOntoService ontology where the requestor can retrieve any detail about the

available services using SPARQL queries. Domain expert developers can also add specific rule to differentiate between services offering same data type (like QoS and QoS parameters).

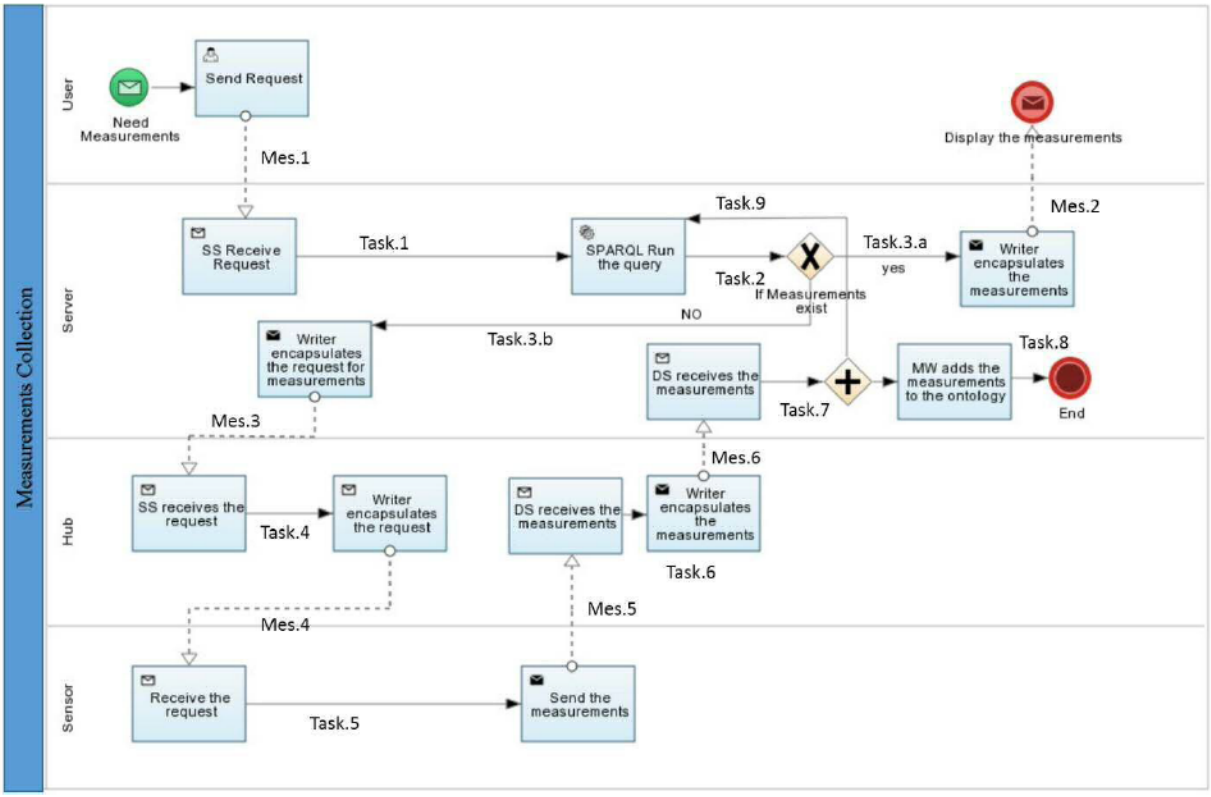

Figure 4.16-Measurements Collection

When a user requests a service, for example the heart rate of a patient, the Service Data Scanner is invoked to retrieve the requested data. Then the SPARQL agent is called to run the SPARQL query for service details retrieval. The service profile will permit to the user/developer to choose one service profile between many service profiles describing the same service. The service process describes how the service processing phase will be done. Finally, the service grounding permits to invoke the adequate writer agent. Going back to the previous example, let's say that when retrieving the heart rate we have the CoAP and HTTP protocols in the Service grounding. And let's also consider that the node requesting this heart rate measurement does not implement the CoAP protocol. Then, the HTTP writer will be invoked to send the answer to the client.

4.3.2.5 Service Processing Phase

The service processing phase depends on the service process description where input/output parameters, precondition, post-condition, method and effects are described. For each service, the developer will define a Web Service/agent in order to execute the adequate process (Service Processing Agent). But, before execution, the service should be granted by the Traffic Management Agent to ensure that the urgent data have the highest priority. For example, when a user requests the heart rate of a patient, the service discovering phase is executed in order to pass the adequate data (e.g. requester's email to verify the user's permission to access the data, the protocol of communication, the time of the requested data, the value from the measurement class, etc.) to the service processing that processes the data and gives the response to the

requestor after calling the writer. Figure 4.17 depicts the activity diagram of the service processing phase. Based on the service profile, the user should be authenticated (if needed) and the QoS (or other parameters like QoI) is retrieved. The I/O parameters are given to the Service Processing agent to process the requested service (in some cases the processing methods can be retrieved from the MyOntoServiceProcess ontology to call the corresponding function). Once the output is generated, and the traffic management grants the transmission of the result, the suitable writer (based on the service grounding) is invoked to send the result to the Application Agent.

4.3.2.6 Network Management

Few existing architectures provided the possibility for an interoperable solution for WSN remote management, regardless the network, data link and physical layers used in the deployed WSN. One of our contributions is this possibility provided by our proposed architecture. Let's say that we need to automatically change the coordinator when its battery level is smaller than a threshold. The WSN's manager can enter this information via a Graphical User Interface (GUI) interface. The Service Scanner retrieves this information, and the rule agent is invoked to translate this constraint into a SWRL rule. After that, the inference engine verifies that the ontology is still consistent and notifies the WSN's manager using the same GUI system via the writer. If the rule is added, the trigger agent can be invoked when a coordinator has been modified, which, in turn invoke the adequate notification agent to notify the WSN's manager that a new coordinator has been set.

Many more scenarios can be envisioned. It is up to the developer to choose what he needs, starting from just monitoring the WSN's characteristics (SPARQL queries), passing through modifying the characteristics of certain nodes (e.g. the current mode from active to sleep), till the automatic change in the WSN based on domain experts 'requirements (SWRL rule).

4.3.3 Summary

In summary, this architecture permits the automatic nodes discovery, services discovery, services matching, remote monitoring, and automatic notification, with the minimum user and developer interaction. It masks the heterogeneity between different nodes and communication protocol due to the use of scanner agents and writer agents. In addition, it enables the re-use and sharing of the information between different application/service providers due to the use of SPARQL queries and service scanners. Furthermore, it opens the door to new creative solution for domain expert developers in order to enhance data fusion and decision making using SWRL agents.

This proposed architecture can be used for small flat WSN to large scale WSN where the management of nodes and the data fusion are mandatory in such fields. To validate our architecture, as well as to clarify its functionality, a detailed implementation of the smart home for elderly' fall monitoring is provided in the next section.

4.4 Smart Home for Elderly's fall monitoring

As mentioned previously, being part of the ETSI TCSmartBAN working item 1.1 pushes us to choose a biomedical application. Moreover, multi-tier system architecture has been envisioned to clarify how to use our proposed architecture.

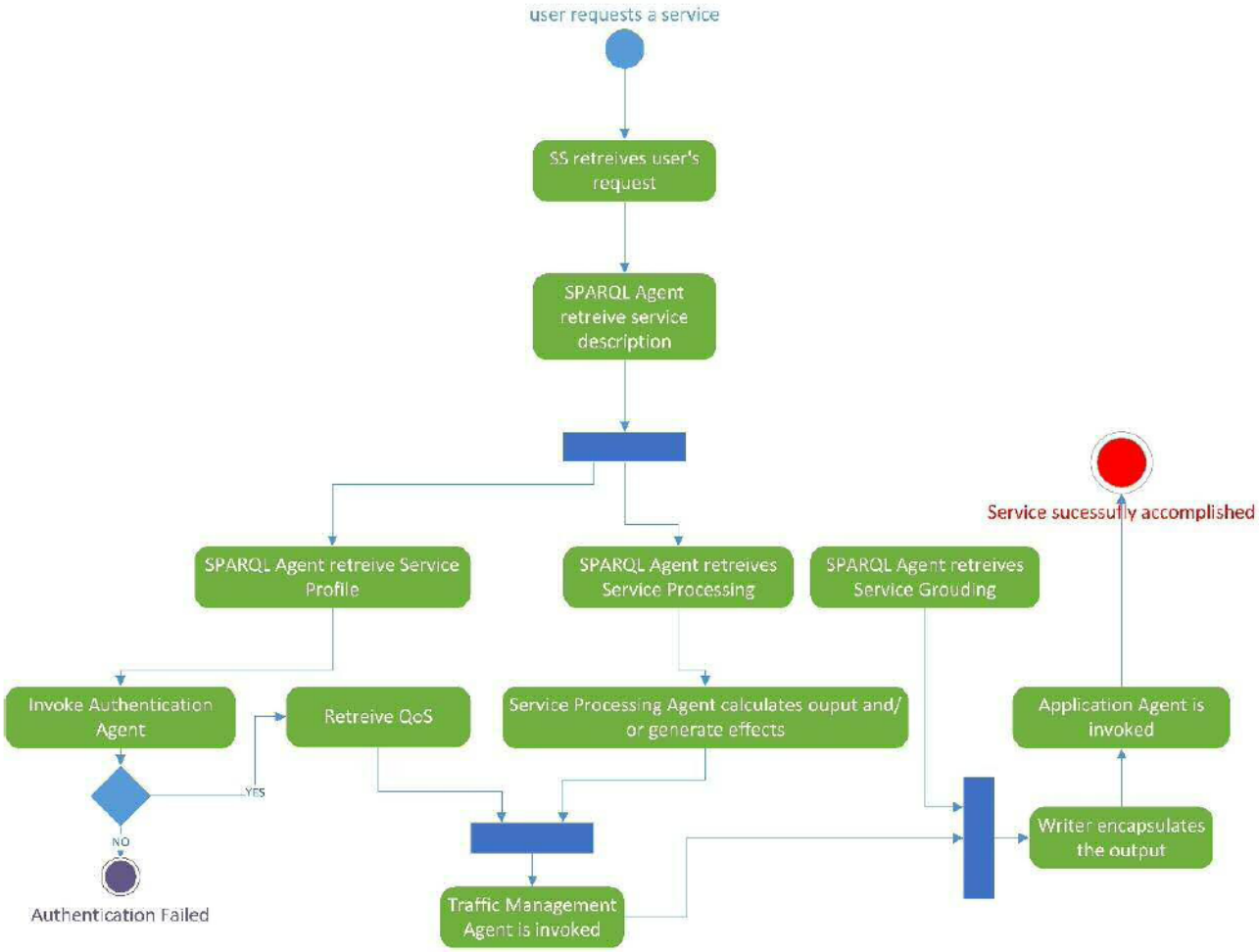

Figure 4.17-Service Processing Phase

As briefly described in Chapter 3, a smart home for elderly fall monitoring system has been conceived, the OntoSmartHome system. The system consists of:

1- BAN: the body area network containing the HR sensor (in our use case the H7Polar Sensor); Note that more biomedical sensors can be used without the need to reconfigure the application like temperature sensor, SPO_2, etc.

2- PAN: the personal area network consisting of :

- The BAN
- The smart phone or PDA which plays the role of accelerometer sensor, ambient light intensity sensor, ambient temperature, ambient humidity and home location sensor. Furthermore, the smart phone can process, reason and display data. It can be used for alarm generation, remainder and alerts.
- The Bluetooth Location sensors placed at different rooms in the home to identify the indoor location of the patient.

3- The LAN : the local area network that includes all the components at home :

   - The PAN; It can contain more than one PAN if more than one patient is monitored at the same home.
   - The local semantic server where the heavy treatment of data is done.

4- The medical, clinical, emergency, authentication or any external required server connected to the LAN via the Internet.

5- The relatives terminals who are allowed to monitor the patient's system as well as interacting with it.

Note that our implementation is only a prototype of what the system can encompass. Many scenarios can be derived, and we will propose these scenarios even though we could not implement them all. First of all, let us describe our implementation assumptions.

4.4.1 Implementation assumptions:

Patients are equipped with H7Polar sensors for measuring their heart rate (any type of heart rate sensor can be used). Bluetooth LE beacons are used to identify the indoor localization of the patient. It consists of many fixed position Bluetooth beacons distributed all over the house segmented as zones or rooms where the indoor localization process should identify in which of these zones the patient is. The Smartphone is considered as the receiver in this subsystem where it is supposed to measure signal strength RSSI of all Bluetooth beacons and to determine the location accordingly using Bluetooth. A calibration phase, when setting up the system, is required to identify the range of RSSI for each zone. In fact, we used the Smartphone for the indoor localization calculation because it has a Bluetooth LE interface while our used local sever does not have one. But, without the need to modify anything in the overall architecture, this calculation can be conducted into the local server, if it is equipped with a Bluetooth LE interface. Just as indoor localization sensors, any type of sensors can be used and different methods of calibration can be chosen by the system's developer.

For fall detection, the three axes accelerometer sensor built in the mobile is used. The measurements of this accelerometer sensor also permit to identify the posture of the patient (see section 3.5). It is important to remind here that the main purpose of this prototype is to show how different agents interact for automatic node discovery and service discovery. More sensors can be added to give highest reliability in fall detection. We limited our choice of acceleration sensors to those built in the Smartphone, just for availability. It would have been obviously more efficient if dedicated linear and angular acceleration sensors would have been used, thus providing a smart phone independent system with highest reliability and persistence. Once a fall is detected, the application asks for the elderly to respond. If he/she does not respond (by pressing a button) within 1 minute, the system sends an alert message (SMS or notification) from the application to the relative's phone. If the elderly person turns off the alarm, nothing will be sent. Figure 4.18 depicts the flow chart of the falling detection process.

Only an Android Mobile version has been implemented for testing purposes. Moreover, the authentication relies on the Google API authentication solution (this was only retained for implementation simplicity purposes; security is out of the scope of our testing case).

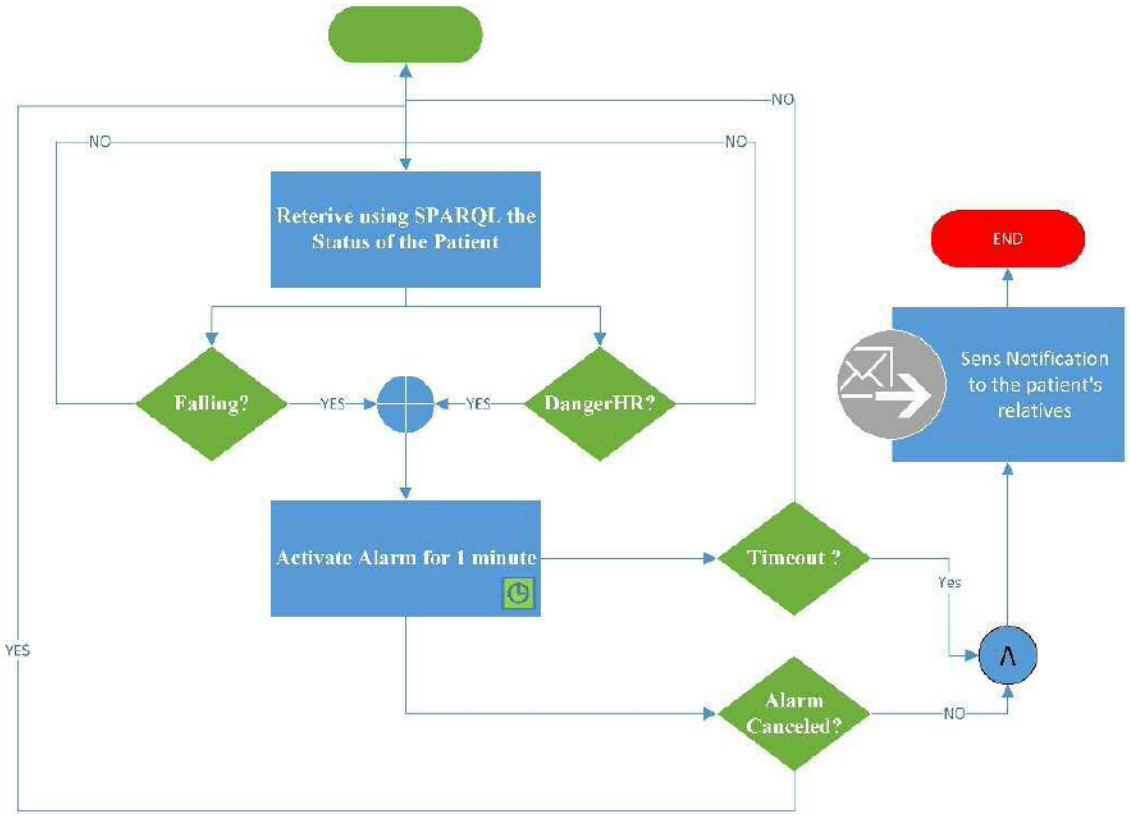

Figure 4.18-Flow Chart of the Fall Detection

For data publishing, the CoAP protocol is used to provide the patient's posture, the patient's location, the patient's heart rate, the ambient temperature and the ambient humidity.

Because our system aims to provide the monitored data to outside users (relatives and medical staff) and not for the patient himself, the semantic layer is implemented in the local server only. However, semantic treatments can be projected to the Smartphone in order to infer, for example, the status of the patient directly from his/her PDA. In this case, only the MyOntoSensProcess ontology should be modeled in the PDA and the status can be inferred from the accelerometer measurements. The AndroJena could be used to enable semantic processing on the mobile side.

Therefore, our data provision layer consists of the sensors and the patient. Bluetooth LE scanner agent (to collect data from sensors) and JSON scanner and writer agents (to receive/send data from/to the local server) are installed on the smart phone.

Regarding the local server, the following agents should be implemented:

- JSON scanner and writer agents (to receive/send data from/to the patient's Smartphone)
- Node semantic wrappers (NW), process semantic wrappers (PW), service semantic wrappers (SW, to translate raw data to semantic information)
- Pellet reasoner agent (to infer semantic data)
- SPARQL agent (to retrieve semantic information)

On the cloud monitoring server, the following agents should be installed:

- JSON scanner and writer agents (to receive/send data from/to the local server, as well as for sending data to remote medical servers if needed),
- WSN semantic wrappers (WW), Node semantic wrappers (NW), Process semantic wrappers (PW), Service semantic wrappers (SW, to translate raw data to semantic information),
- Pellet reasoner agent (to infer semantic data),
- SPARQL agent (to retrieve semantic information residing on the cloud monitoring server or on the local server),
- CoAP server agent to create resources (patient's information) and to respond to GET request from patient's relative and medical staff users,
- Authentication agent to authenticate users' Smartphone (we limited our choice to Google API authentication server for implementation simplicity purposes and because we developed Android mobile application. Security is out of the scope of our testing case and more advanced authentication servers can always be implemented latter on).
- Notification agent to notify medical staff and patient's relatives if the patient has fallen or if his heart rate exceeds the maximum normal value (we used the goggle GCM authentication server due to its easy integration but other notification servers like MQTT can be used).
- Traffic Management agent to prioritize notification data flow.

Now that the mobile application has been installed on the patient's smart phone and his local server is equipped with all required agents (more scanner/writer agents can be added if other types of communication protocols are used like ZigBee, RFID, etc.), the patient is now ready to use the system.

4.4.2 System' phases

First, the user has to accomplish the setup phase, then the application will be in data collection phase where measurements are collected and sent to the local server. The server then sends the required information to the cloud monitoring server. The cloud monitoring server will: authenticate external users, provide requested data, and send notifications when necessary.

4.4.2.1 Setup Phase

The setup phase consists of:

1- WSN and Patient initialization: where the creation of the WSN, Cluster, and Contact individuals should be setup in MyOntoSensWSN ontology

2- Node discovery: where the system detects the presence of the sensor/node, discovers its characteristics and adds these information to MyOntoSensNode and MyOntoSensProcess ontologies.

3- Node management: if additional management are needed. In our case, the Indoor Localization demands a calibration phase to determine the patient's location.

Initialization

When the patient uses for the first time the application, he/she can either choose to join an existing WSN, or create a new WSN. If a new WSN should be created, JSON Writer (JW) agent sends the mobile IP to the local server. Then, the local server decapsulates the JSON message, generates a new WSNID and sends it to the patient's mobile, and finally invokes the WW (WSN Writer) to crate the new WSN individual. Else, the user should enter the *WSNID*. In both cases, the server responds with the new created *PatientID* that will be displayed on the home screen. Figure 4.18 depicts the flow of messages during the initialization phase. The main player in this phase is the WW deployed on the cloud server which creates (refer to Chapter 3 section 3.5.2):

- New WSN individual
- New Patient individual
- Two clusters: "MobileCluster" and "LocalCluster".
- "hasOwner" between the WSN and the Patient
- "isFormedof" between the WSN and each Cluster

This semantic information is only created on the cloud server (in the message 5.a in Figure 4.19) because it is needed to manage different WSNs and to responds to external user's request. This can be done due to the modularity of our proposed ontology where each node can use the needed module.

The other player is the Node Wrapper (NW), deployed on the local server. It creates:

- Two Hubs: "MobilePhone" and "LocalServer".
- For each node, the WIFI interface (because the data was received form the phone via WIFI Scanner).
- "ElectHub" between "MobileCluster" and "MobilePhone", as well as between "LocalCluster" and "LocalServer".

These semantic information will be first created on the local server (Figure 4.19, message 3.b), and then created on the cloud server using Insert SPARQL query. Note that in message 1, as well as in message 2.b of Figure 4.19, more information can be retrieved from the Smartphone and the server respectively, such as the manufacturer ID, the operation system, the RAM capability, etc. In other term, the properties described in "NodeType" and "Node" classes of MyOntoSensNode ontology can be retrieved from the Smartphone. When the initialization phase is completed, the user should click the "setup WSN" button in order to begin the setup process, starting with the login activity using Google + API.

Log in Phase

When this activity loads, the user is directly connected to the Google server via his/her Gmail account. The application can transparently retrieve some information directly from his/her account without any interaction like his/her mail, first name, last name and gender. In addition, the application displays a user Interface for the patient in order to fill other required fields for describing the Patient class in MyOntoSensWSN ontology (in our application: weight, height,

gender, location, email, mobile phone). The JSON Writer (JW) on the smart phone sends this information to the local server. The JSON Scanner (JS) reads this information and invokes the Patient Wrapper (PaW) for semantic annotation.

After the login phase, the WSN number, the patient number, and the patient's personal information of the Login activity should be stored on the Smartphone. Or, shared Preferences is an API from Android to store and retrieve applications that will persist across user sessions, even if the user stops, exits or kills the applications. Accordingly, the "Shared Preferences" solution was adopted as the optimal solution for persistent storage.

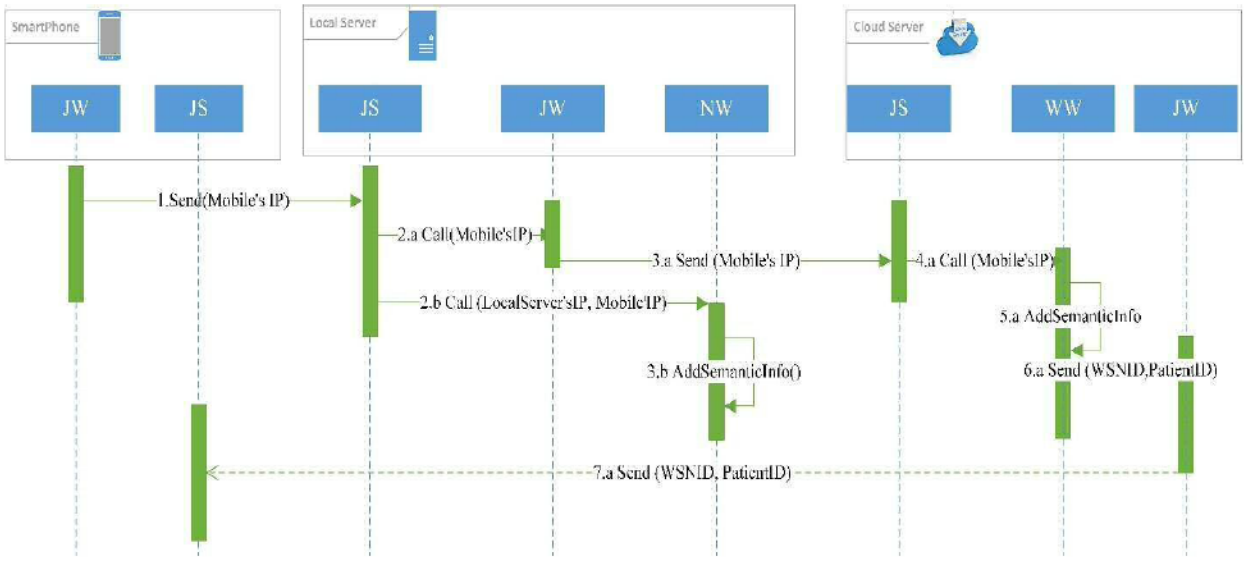

Figure 4.19-Flow of Messages in Smart Home Initialization Phase

4.4.2.2 Sensor Discovery Phase

Two types of sensors exist in our use case, the Bluetooth LE sensors and the built-in mobile sensors. It is very important to highlight that, only for availability reasons, Bluetooth LE sensors are detected by the smart phone because our server was not equipped with Bluetooth LE interface. Nothing will change in the general architecture if it is the task of server to discover those sensors. We should only implement BLE scanner (BLES) on the server side. In our case, the BLES on the mobile side will detect the sensors and, based on the UUID; it can discover the sensor characteristics. The service type, device MAC address, firmware version, hardware version, model number, software revision, and battery level are firstly detected in order to discover the sensor characteristics. The BLES invokes the JW to send this information to the local server. The JS on the server side reads this information and invokes the NW and the PW to add the semantic annotation. This flow of messages is depicted in Figure 4.20 .This is what exactly happens with H7Polar sensor. Nevertheless, the Rad Beacon used for Indoor localization needs additional set up phase to be discovered properly because the UUID does not reflect the intend service (Indoor Localization).We just want to locate the patient in a certain zone or room. For that purpose, once the Rad Beacons are discovered, they will be displayed on the patient's mobile screen in order to be selected by the Patient. To enhance the selection process, the developer can assign, in addition to the UUID, Major and Minor parameters describing the main

functionality of the beacon (e.g. the location). After submitting his/her choice, the JW sends the beacons' MAC address to the cloud server. The NW adds these nodes to MyOntoSensNode ontology. Then, the user specifies the number of rooms in the house. For each room, he/she enters its name and an initiation phase is launched for 20 sec to read values from all registered beacons. After 20 sec, the minimum and maximum RSSI values read from each beacon are registered for this room in order to locate the patient. The JW sends to the local server the rooms' name. Then, the PW add these names as data for the Indoor Localization process. So, the flow of messages shown in Figure 4.20 will start after the initial localization phase.

It is time now to discover the built in sensors. The JW on the mobile side sends JSON message having as title "Sensor" and containing the sensor's description (Process: acceleration, data rate: 50 samples per second). Note that in our specific use case, after drawing the fall detection graph on data rate variation, we are taking 50 samples per second. Hence, the sensors are discovered and the local server sends an insert SPARQL query to cloud server to add these semantic information. Figure 4.20 depicts the flow of messages for automatic H7Polar BLE sensor discovery. This flow of messages remain the same for all BLE sensors that do not need any additional setup phase (e.g. if we are using a BLE body temperature sensor). Note that the same messages shown in Figure 4.19 will be retained for Built-in sensors, but the JS will detect their presence instead of the BLES.

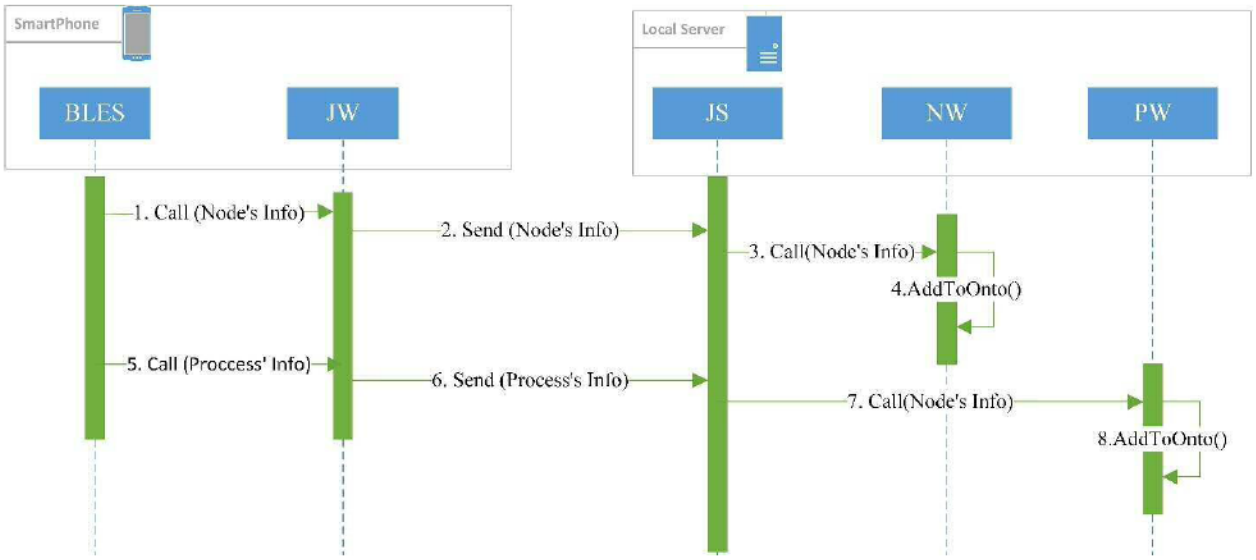

Figure 4.20-Sensor Discovery in Smart Home Scenario

4.4.2.3 Measurements Collection Phase

After the initialization phases, the home page activity starts and from now on this will be the User Interface (UI) page of this app. Each time this activity is loaded, on one hand, the BLES identifies the connected BLE devices and sent it to the servers. The local server runs a SPARQL query to verify if these nodes already exist, otherwise they will be added as shown in Figure 4.20. In this way, the system is always updated transparently, without the need of any intervention form the user. This feature provides a plug and play mechanism for sensors enablement. On the other hand, because our main aim is to detect the fall of a patient, the

Smartphone verifies if the beacons are powered (the Indoor Localization is mandatory in elderly fall detection). Otherwise, a notification will be displayed telling the patient to power on the beacons. If he did not power those beacons after 20 minutes, a message is automatically sent to the system's support to assist the elderly (maybe he/she is not able to do so). Moreover, if the mobile battery level is smaller than a threshold, a notification is displayed to charge the mobile phone. These precautions are done to be sure that there is enough power resources to detect the fall of the elderly. From now on, the data collection phase has started. The measurements are retrieved form the sensors, sent to the local server, which in turns adds it to the ontology. The local server will send these measurements to the CoAP cloud server in order to update the resources value. The JSON Writer (JW) on the server side sends a JSON message to the Cloud server having as title "Measurement", and containing the "WSNID", "PatientID" and the process (HR, Humidity, Ambient Temperature, Actual Location, Lightness, and the three axes linear acceleration) with the value (e.g. "HR": 117). The Cloud server sends a POST CoAP message to update the measured value. Figure 4.21 depicts the resources hierarchy (URI) of the CoAP Server. The patient's posture will be inferred from the SWRL rules (see Chapter 3 section 2.5.2 Rules 3.13), queried using SPARQL query 4.1, and then posted to the CoAP server. The patient's status is left to the patient to be filled in his/her mobile application in order to inform the relatives about his status (like in social media application, it could be an emotional status). More works can be done regarding this point, for example link this status to Facebook application or others.

SPARQL Query 4.1 SELECT ?Status WHERE { ?Patient smart:hasStatus ?Status. Filter (?Patient = smart:WSN1_Patient2)

Note that WSN1_Patient2 parameter will be changed according to the monitored patient.

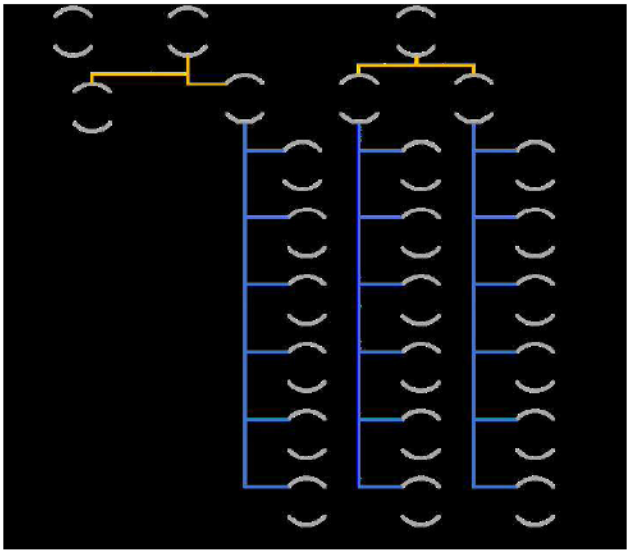

Figure 4.21-CoAP Resource' Hierarchy

Moreover, relatives should be added from this activity. After pressing the menu button, the patient can add Relatives to his/her WSN. The JSON Wrapper (JW) sends the relative's name, email, phone and the relations (parent, sister, etc.). The Contact Wrapper (CW) is invoked to add a new contact to MyOntoSensWSN ontology, and to add *isRelativeOf* relation between the patient and the created contact.

4.4.2.4 Service Description Phase

All processes (HR, temperature, Indoor Localization, etc.), as explained in Chapter 3, are automatically inferred as atomic Process. By default, the WSN will be engaged in these services. These services are described due to SWRL rules already given in Chapter 3. In particular, for this implementation, the services could be accessed using the CoAP protocol. Therefore, the service grounding for these aforementioned services will be the CoAP. For simple services, it is the duty of the Local server to create it. For our special cases, SWRL rules are added in order to describe these services (see Chapter 3), e.g. *MaxHR* service. Moreover, a composite service *FallDetection* is created. These two services have, as effect, a notification, and thus the notification agent is invoked.

4.4.2.5 Service Request Phase

When the relative runs the application for the first time, he/she should specify the Gmail of the monitored patient. The JSON Wrapper (JW) on the relative's mobile sends the patient's Gmail and the relative's Gmail to the cloud server. The JSON Scanner (JS) on the cloud server reads the request and invokes the SPARQL agent that runs the Query 4.3 (the emails should be changed according to the patient and relative emails) to retrieve the WSNID.

SPARQL Query 4.3

Select ?WSN where{ ?WSN wsn:hasOwner ?Patient. ?Patient wsn:isRelativeof ?Contact. ?Patient rdf:type wsn:Patient; wsn:email ?email; Filter regex(?email, "lina_nachabe@yahoo.com"). ?Contact rdf:type wsn:Person; wsn:email ?Contact_Email; Filter regex (?Contact_Email, "sahar_assaf@gmail.com").}

If the query answer is empty, it means that this relative is not granted as patient's relative and, therefore, cannot monitor the measurements. If the relative has the permission, CoAP GET requests are sent to the cloud server in order to retrieve the measurements. Finally, the measurements will be displayed on the home screen. Figure 4.22 depicts the process of retrieving the measurements on the relative's mobile application.

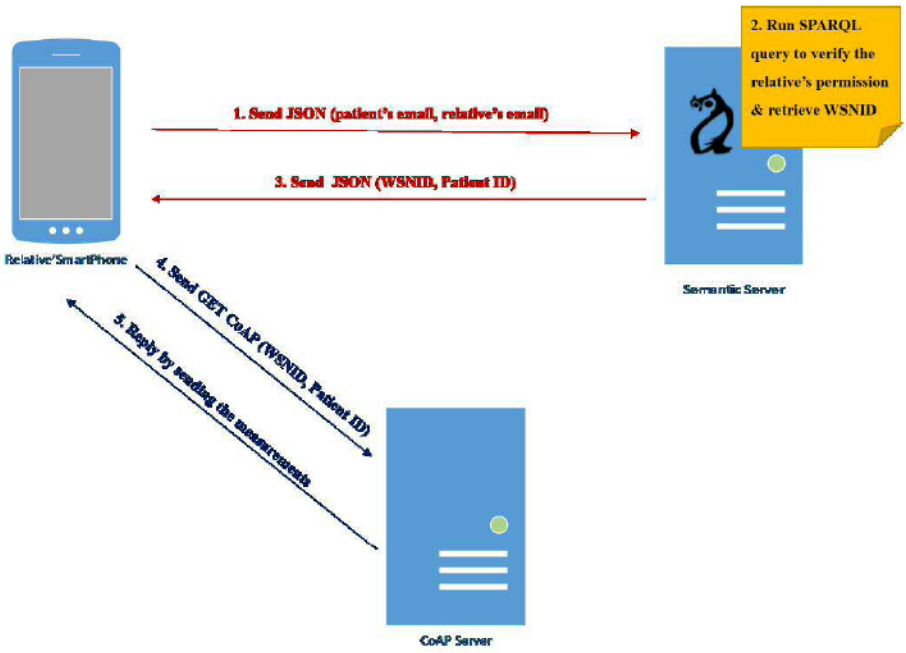

Figure 4.22-The process of getting measurements on the relative's application

4.4.2.6 Notification Phase

Periodically, the local server sends the SPARQL Query 4.4 to retrieve the status of the patient. If the status is *Falling*, or *DangerHR*, the notification agent is invoked. We used in our implementation the Google Cloud messaging (GCM) notification agent (other notification agents can be deployed without affecting our overall notification process). GCM main advantage is the avoidance of the polling mechanism between app and server. When a particular server has data to be sent to a group of users of a specific app, it sends the data to the GCM server that pushes the notification to all users' app. It is based on three main components:

- Application Server
- GCM Server
- Android phone

Its main parameters are:

- Registration ID: the GCM server generates a unique registration ID to each android app running on a single device that allows it to receive notifications.
- Sender Authentication Token: it is an API key saved on the application Server that gives it authorization to access Google services. The API key is included in the header of POST requests that sends messages.
- Sender ID: it is a project ID you acquire from the API console. It is used in the registration phase to identify an android application that has permission to send messages to the device.

How the notifications are generated and received?

First of all, we registered our cloud server (in this phase Application server) to the GCM notification server. In response, the GCM server sends to the application server the ServerID to be used later on each time a notification should be sent. Second, on the relative's application and after logging in, the user will be asked to register to the notification server. The user sends an activation request via the mobile application to register in the notification service of this app. This request should include the Project Number (Server ID dedicated for our application and set statically in the code). The GCM server sends the Registration ID (RegID) to the client as a response of the user's request. Hence, the user sends its RegID and email to the application server using JSON message. The application server stores the email and the RegID for each relative in a local database. From this point, the notification process is enabled. The semantic cloud server sends SPARQL (Query 4.4) periodically, to check if a patient is falling, or if his heart rate exceeded the normal value.

SPARQL Query 4.4 Select ?Contact_Email where{ ?WSN wsn:hasOwner ?Patient. {?Patient smart:hasStatus smart:Falling} UNION {?Patient smart:hasStatus smart:DangerHR}. ?Patient wsn:isRelativeof ?Contact. ?Contact rdf:type wsn:Person; wsn:email ?Contact_Email; }

For those patients in danger, it will retrieve their email and send it to the notification application server, which in its turn, retrieve the RegIDs and send it to the GCM server. The key point behind separating the role of semantic cloud server and notification server is the modularity where each server can be on a separated machine. Moreover, advanced security mechanism can be performed on the notification server where each developer can conceive its own strategy for notifications (by call, by SMS, etc.). Figure 4.23 depicts the notification process.

4.4.3 Tests and Results

The applications were tested on Samsung GALAXY S4 (Android 4.4.2 KITKAT-API 19 and Android 5.0.1-API 21 Lollipop after upgrade) and SONY XPERIA SL (Android 4.1.2 Jelly Bean- API 16). We first considered one patient per WSN. We tried to install the application on two different patients' mobile. For each patient, one relative was added. Then, we considered the scenario where two patients are sharing the same WSN, i.e. sharing the same Rad Beacons for the indoor localization. The distance between rooms were about 20 m and three Rad beacons were implemented in three different zones at the patient's home. We will illustrate only one patient's screen shuts where he/she is adding a new WSN.

For testing only, the cloud server and the local server were on the same machine, but working on different port numbers. It took about 20 seconds to load the full ontology. It uses 425 692 Kbits of RAM and 76% of the CPU as maximum values. Unfortunately, the Pellet in Jena does not support "subtractDayTimeDurations" expression, thus the status "Falling" could not be retrieved.

After loading the ontology, the server tries to classify it. We noticed that the server is going in an endless loop and cannot classify the individuals due to the use of "abs" expression. Or, this expression is used in all the rules related to the "Status". When removing these rules, the ontology was well classified within 4 minutes. Thus, we decided to calculate the status using a "Status Agent" that receives the acceleration values from the JSON Writer (JW) and calculates the patient's status. Then, it invokes the CoAP Writer to send the value to the CoAP server. In case the user is falling, the notification agent will be invoked.

After verifying that the servers has classified the ontology, we started by testing our mobile applications. When opening the patient's application for the first time, Figure 4.24 appears and the patient chooses to create a new WSN. Then, the server assigns new numbers for the created WSN and Patient. Therefore, the server created these individuals (Figure 4.25).

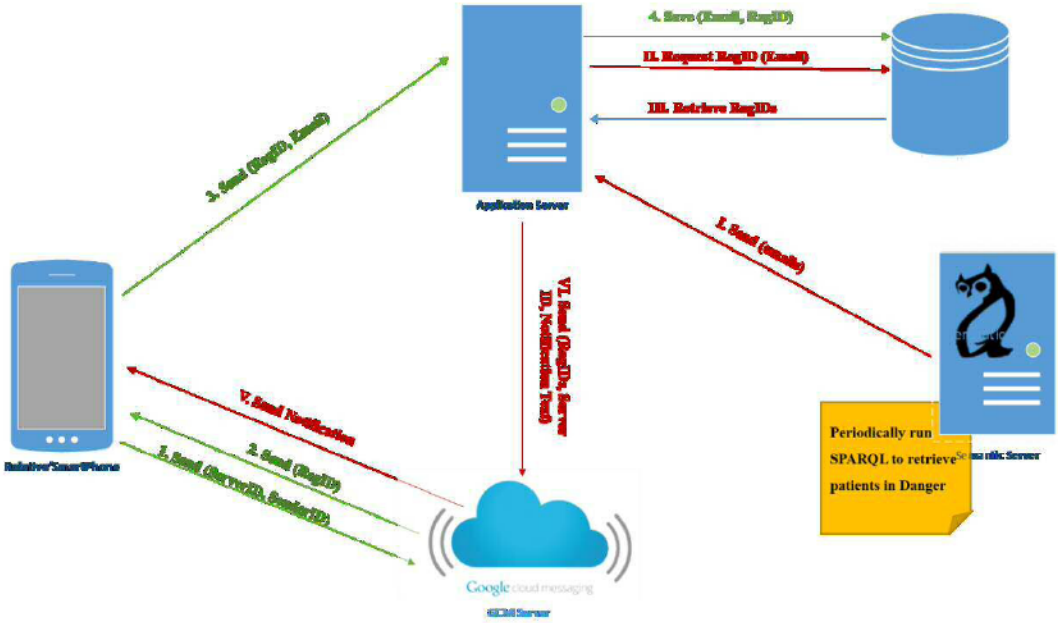

Figure 4.23-Notification Process

Figure 4.24- First Page of OntoSmartHome application

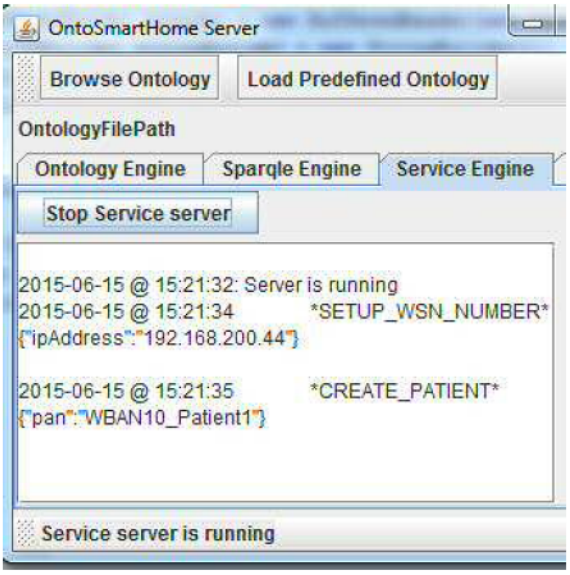

Figure 4.25-Creating a new WSN and a new Patient

Figure 5.26 depicts some statements added by the server once it detects the patient's mobile phone.

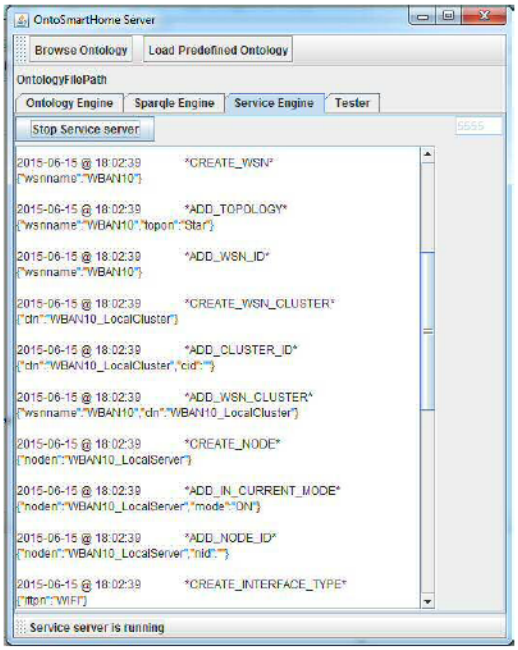

Figure 4.26-Creating Clusters and Nodes

After creating a new WSN or adding the patient to an existing WSN, the user should click the "setup WSN" button in order to begin the setup process, starting with the login activity using Google + API. When the login activity is loaded, the user is directly connected to the Google server via his Gmail. The application can then retrieve some information directly from his account (without his interaction) like, e.g., his mail, first name, last name and gender. The application also provides the patient an interface, depicted in Figure 4.27, to fill other required field for the Ontology.

Figure 4.27-OntoSmartHome Login Page

After a successful signing in, the patient is prompt to the indoor localization setup phase where he/she should precise the number of rooms and do the calibration.

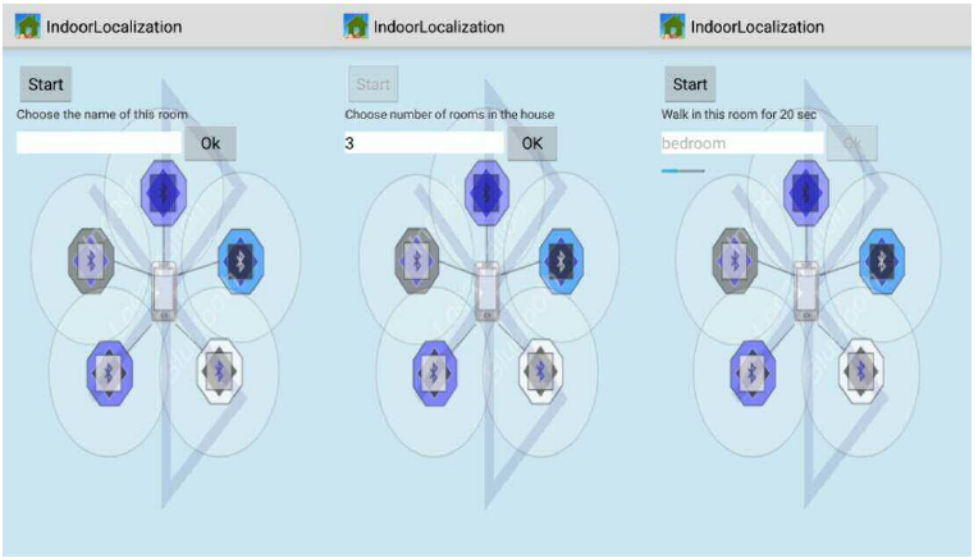

Figure 4.28- OntoSmartHome Indoor Localization

Figure 4.29 depicts the process of creating individuals related to the indoor localization.

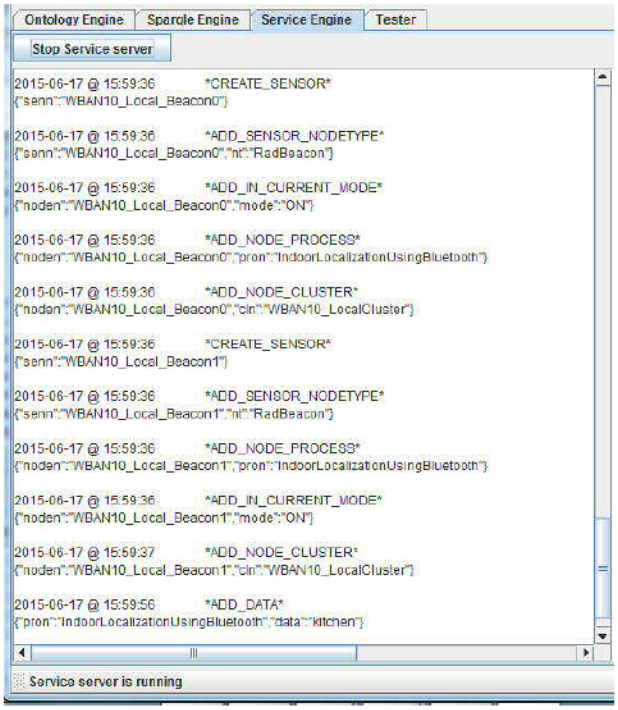

Figure 4.29- Server processing for creating Indoor Localization's individuals

After finishing the calibration phase for all the rooms, the heart rate sensor will be detected. The user should confirm its usage. Figure 4.30 depicts the H7 Polar discovery.

Now that when all the sensors are successfully discovered, the patient will be driven to the home page depicted in Figure 4.31. This page is the same for the patient and the relatives. It is the page where all the data are get from the CoAP server and displayed on the screen. To this point, all

individuals are created and well classified on the local server, as well as the cloud server. Figures 4.31, 4.32, 4.33, and 4.44 depict some created and inferred information.

Note that the user can add a new beacon for the indoor localization at any time. For that purpose, we went to the setup *IndoorLocalization* option, we added a new beacon, we calibrate it and it was successfully discovered. Moreover, we removed the H7 Polar sensor, thus, the server received from the JSON Writer (JW) that the H7 sensor is off. So, when running, the node discovery query, the H7 sensor is obviously not discovered anymore.

Relatives can be added at any time by accessing *AddRelatives* option. Figure 4.45 depicts the page where the patient is adding relatives. The relatives should login using their Gmail account when they install the application for the first time. Afterwards, the home page (Figure 4.31) will appear. The values will be requested every two minutes. We noted that the humidity and the temperature did not vary a lot. But, the luminosity varies each 3 minutes. For testing, we installed "Polar Beat" application to compare the heart rate displayed on the relatives' smart phone and that displayed by the installed application. The two values were the same.

We tried different scenarios for fall detection. Falling after standing, from bed or from a chair were successfully detected.

The patient's application was tested for about 1 hour, the battery consumption was about 5%, and the average CPU usage was 16%.

In summary, although this system does not provide the best solution for fall detection, it offers the flexibility to enhance this detection whenever needed by just adding the necessary sensors. The nodes are discovered within 20 seconds, as well as all basic services. On the relatives' side the values were displayed in real time.

Another possible solution is to use MyOntoSensProcess ontology on the Smartphone. This can be made using AndroJena. However, it doesn't support Pellet reasoner neither SPARQL queries. To get the maximum inference the "OntModelSpec.OWL_MEM_MICRO_RULE_INF" reasoning model should be used. It can infer inverse properties, sub classes and sub properties used in MyOntoSensProcess. In our scenario, this annotation is not useful because we are trying to minimize the work of the Smartphone. However, this annotation seems useful in large scale networks where the sink can be equipped with a microprocessor that annotates the data before sending it to the gateway. More researches can be conducted on this point.

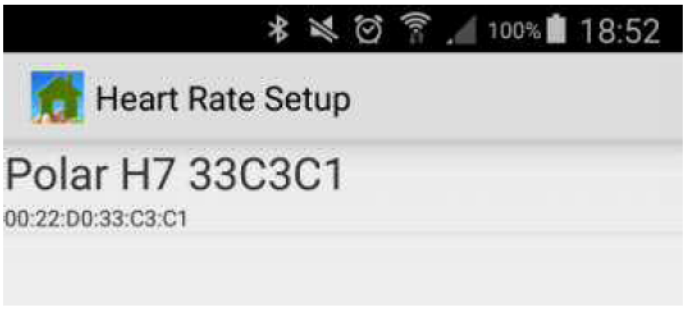

Figure 4.30-H7Polar discovery

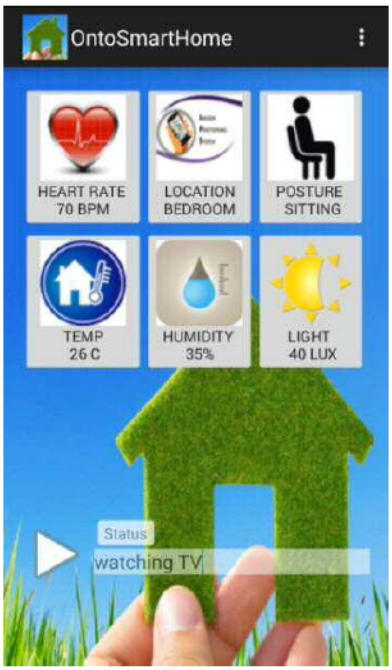

Figure 4.31-OntoSmartHome home page

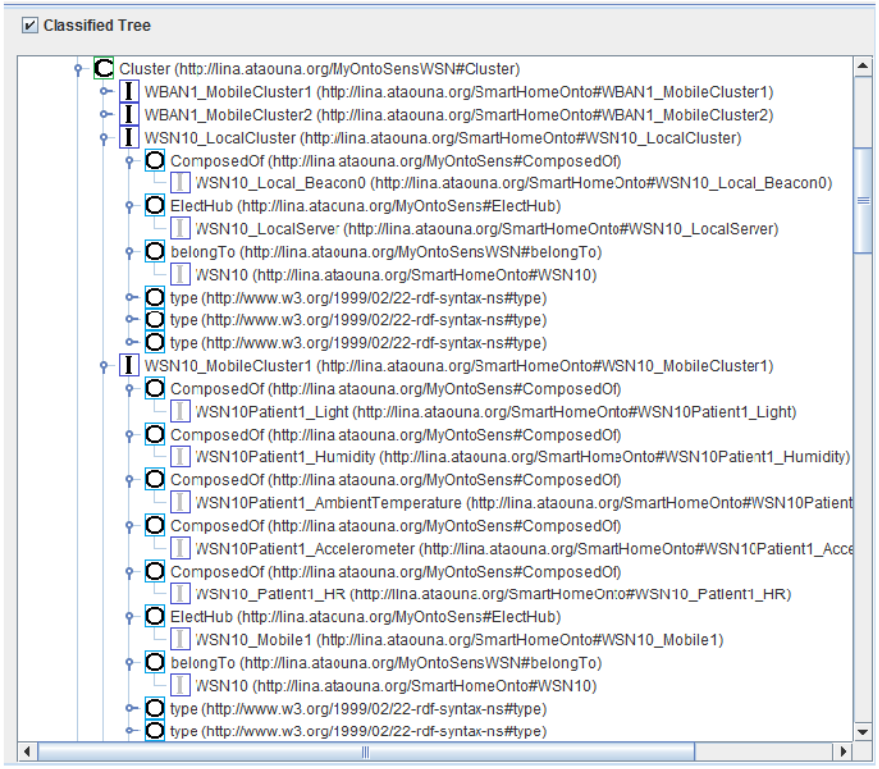

Figure 4.32-Clusters Individuals & Properties created on the Local Server

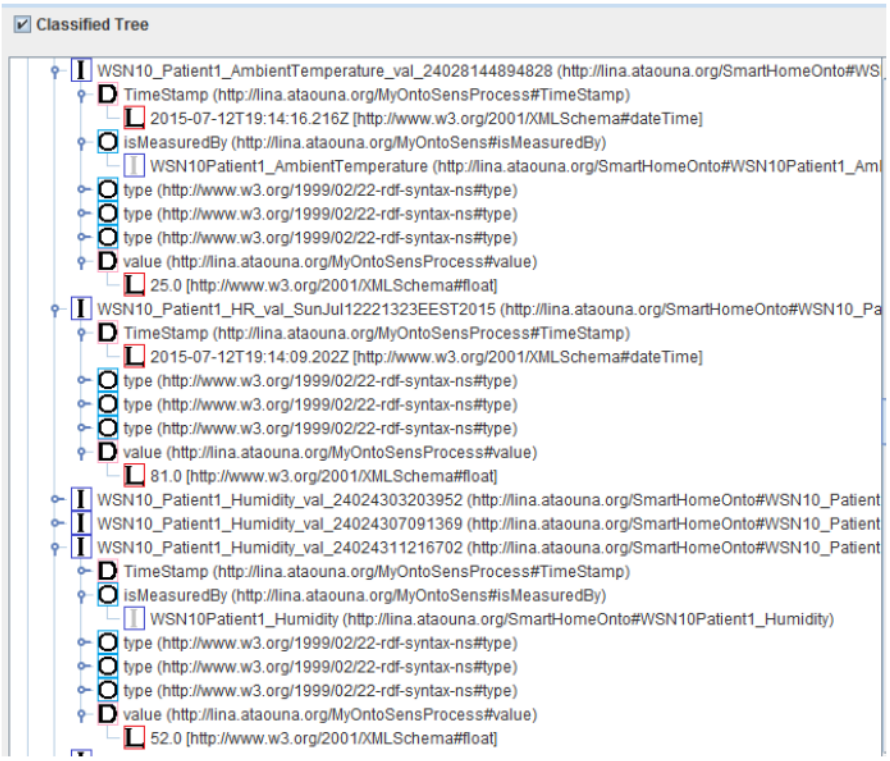

Figure 4.33-Measurements individuals created on the server

Figure 4.34-Nodes Individuals & Properties created on the server

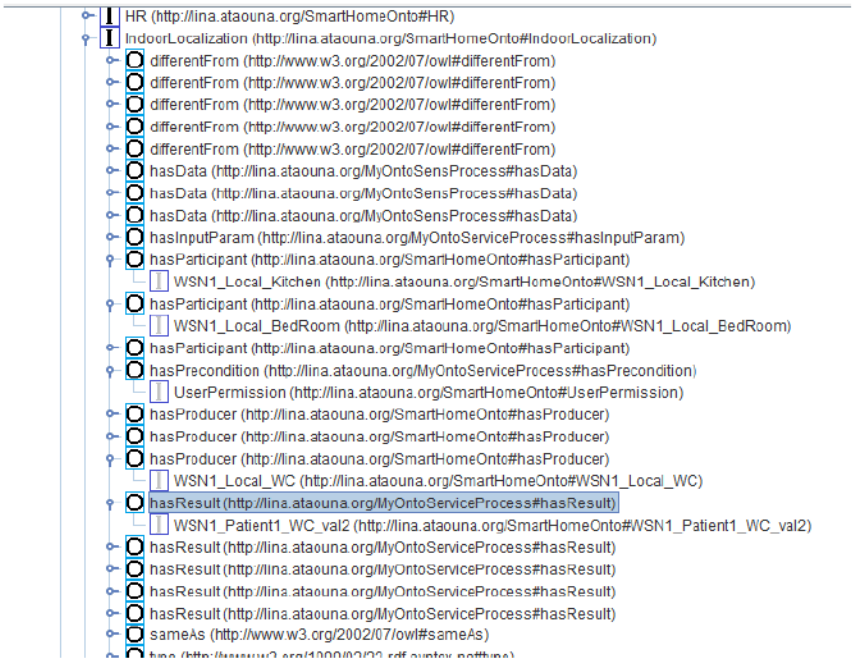

Figure 4.35-Atomic Process individuals created and inferred on the server

CONCLUSION & FUTURE WORKS

Summary

In summary, this thesis mainly addressed the problem of heterogeneity and interoperability in WSN. To tackle these problems the usage of semantic ontologies was primordial.

The first point was to create an open data model that fully describes the WSN, node, process and measurement. This open data model was the fruit of a deep investigation about WSN characteristics and a review on the existing open data model for WSNs as well as the E-health systems. The open data model was formalized using OWL DL language and named MyOntoSens ontology. This ontology was composed of three main ontologies: MyOntoSensWSN, MyOntoSensNode and MyOntoSensProcess, to give more flexibility and lightness where the user can implement the part he/she needs. Almost all the information related to the routing mechanism, reliability, freshness, and security were introduced. To pre-validate the ontology an m-health application has been developed for runners during exercises in order to calculate their calories burned, track their route and inform them about neighbour's runners. This m-health application fully relied on MyOntoSens ontology. It proves that our proposed model is compatible with Bluetooth LE devices. Moreover, it shows that using ontologies and due to inference engine and SWRL, the data sent from the mobile to the server is significantly reduced. Thus the smart phone's battery consumption is decreased.

The second point was to create an upper service model that benefits form MyOntoSens ontology and allows the automatic nodes and services discovery. This upper service ontology should totally describes the services offered by WSNs and enhance the mechanism of choosing the adequate service when requested. Mainly influenced from OWL-S we conceived our upper service ontology: MyOntoService. It is formed of three main ontologies: MyOntoServiceProfile that describes the non-functional characteristics of a service, MyOntoServiceProcess that describes the functional characteristics of the service, and MyOntoServiceGrouding that indicates how the service can be accessed. MyOntoService enhances the mechanism of service matching due to the use of QoS and QOI. In addition it diminishes the confusion in service description due to the service profile properties.

The third part was to conceive an open architecture that relies on MyOntoSens and MyOntoService ontologies. We adopted the idea of MAS to ensure distributed computation, loosely coupled solution and adaptability to environment change. Our proposed architecture is made of four layers:

1- Data provision layer (sensors and external users).
2- Semantic layer where the Data Scanners Agents collect the raw data and send it to the adequate wrapper. It is the role of the wrapper to add semantic annotation based on MyOntoSens Ontology. This annotation masks the heterogeneity problem in WSN and enables the automatic sensor discovery.
3- Service layer where services are created and annotated using Service Wrappers based on MyOntoService Ontology. This annotation helps service developer/user to interact with the WSN regardless the underneath layers.

4- Application layer which is left to the developers and service providers.

To retrieve semantic data (data collection, service discovery, routing information, battery status, etc.), SPARQL Agent is invoked. Additional agents related to traffic management, authentication, security, and notification are conceived. In order to deliver information to external nodes or users the Writers will encapsulate the data based on the adequate grounding (CoAP, JSON, BLE. etc.).

A smart Home to monitor elderly people in their home served as test-bed to validate and evaluate this architecture. It mainly encompasses a LAN (sensors placed in the home and a local server that plays the role of Hub), and a BAN (sensors placed on the body and a smart phone as hub).Two different mobile applications were developed, one for the elderly where he/she can setup the nodes and enter his/her personal information, and one for the patient's relative for remote monitoring. These two applications are adopting the idea of wrappers and writers proposed in our architecture. The local server as well as the cloud server (to monitor al WSNs) are semantic servers where wrappers annotate the information based on MyOntoSens and MyOntoService ontology. In addition to wrappers, scanners and writers, these servers include Inference Agent and SPARQL Agent. The cloud server contains CoAP Server Agent, Authentication Agent and Notification Agent.

Our tests show that our proposed architecture enables the automatic node discovery as well as service discovery within seconds. The classification of the overall ontology took about 4 minutes. The architecture was aware of any change, and nodes were added/ deleted dynamically due to the use of scanners and wrappers.

Moreover, it enhances the data aggregation and provides a flexible way to conceive different services using same collected (from real environment) data (e.g. Maximum Heart rate, Average Heart rate, calories burned from the collected heart rate measurements). It was clear that the amount of data sent from the smart phone to the server was significantly reduced which leads to less smart phone's battery consumption.

Although our test-bed limited the semantic treatment to the servers, Semantic wrappers Agents could be placed on a sink/hub such as a coordinator in ZigBee networks, or smart phones. In large scale network, where many coordinators are implemented, Process Semantic Wrappers can be installed in order to give a significant meaning of the collected data (like the process, the data, the accuracy, etc.). In that way, these data can be easily treated by domain expert servers and shared (using URI) by different systems. The decision making will not be limited to servers, but can be driven to these data collectors. For example some rules related to invalid measurements can be added. If the measurements are invalid, the data collector will not send the measurements to the WSN's gateway, thus, reducing useless data transmission. More investment can be done to prove the efficiency of this architecture in large scale networks.

Due to the use of Service Scanners and Writers, the interaction between external users and the WSN is significantly simplified. The user just asks for what he/she needs, and the system is ready to serve him/her with adequate data. Search Agent can be developed based on SPARQL agents to retrieve any required information.

In conclusion, this proposed architecture can be considered the first step toward a common standard in WSN to resolve the problem of interoperability and heterogeneity. We think that this

common open architecture will simplify the interaction between the external users and the WSN components (which still the nightmare of WSN developers). WSN monitoring, management and control can be done via the agents of the modular architecture.

Future Works

Being part of ETSI TCSmartBAN Working Item 1.1, we are focusing on providing a standard for BANs. For that reason, our primary goal is to formalize this open architecture in a common standard, because, we believe that a common understanding must be ensured on all BAN levels to tackle the problems of interoperability and heterogeneity management.

Moreover, the market of Smart Home systems in France is expressively growing, and is set to become a major market opportunity. Thus, we are aiming to test our open architect in real Smart Home Systems for validation and evaluation purposes.

The implementation of this architecture can be spread to other domain of application such as environmental monitoring, smart grids and vehicle networks. We think that vehicle networks can benefit from the interoperability management between different infrastructures to enhance decision making in such smart environment.

Deeper investigation about power consumption evaluation and metric evaluation can be added. In fact, the real evaluation of ontologies comes from the domain expert users. Therefore, as much as this architecture is implemented in different scenarios, an accurate evaluation will be made.

Moreover, the security techniques that should be adopted in this architecture are still discussion points that should be resolved.

Finally, as hash tag "#" was just a sign used for "pound", and it is now changing the social media world by tagging the text and offering linked data. This sign is enabling the data sharing between large ranges of social media solutions and contributed in the success stories of many businesses. This same "#" used to create resources in ontology can change how people are dealing with WSNs. WSN's nodes and data can become searchable, accessible, and manageable resources over the Internet. With the use of ontologies and open general architectures the integration of WSN's in the virtual world is becoming a reality. However, are people ready to share their private data? Is the actual infrastructure ready to support billions of devices sending data at the same time? These smart systems taking decisions instead of people are facilitating their lives or leading to more lazy persons? Maybe, time will answer….

BIBLIOGRAPHY

1. GYRARD, Amelie. An architecture to aggregate heterogeneous and semantic sensed data. In : *The Semantic Web: Semantics and Big Data*. Springer, 2013. p. 697–701.
2. ASHTON, K. That "Internet of Things" Thing. *RFiD Journal* [online]. 2009. P. 4986. Available from: http://www.itrco.jp/libraries/RFIDjournal-That Internet of Things Thing.pdf\npapers3://publication/uuid/8191C095-0D90-4A17-86B0-550F2F2A6745
3. JIRKA, Simon and PAPER, Discussion. OGC ® Catalogue Services Specification 2 . 0 Extension Package for ebRIM Application Profile : SensorML. . 2010.
4. COMPTON, Michael, BARNAGHI, Payam, BERMUDEZ, Luis, LEFORT, Laurent, LEGGIERI, Myriam, NEUHAUS, Holger, NIKOLOV, Andriy, PAGE, Kevin, PASSANT, Alexandre and SHETH, Amit. The SSN Ontology of the W3C Semantic Sensor Network Incubator Group. . 2011.
5. GOODWIN, Caleb and RUSSOMANNO, David J. An Ontology-Based Sensor Network Prototype Environment. . 2006.
6. GEORGE, Tesseris and GEORGE, Baryannis. OWL-S : Semantic Markup for Web Services. .
7. TARUNA, S, JAIN, Kusum and PUROHIT, G N. Application Domain of Wireless Sensor Network:-A Paradigm in Developed and Developing Countries. . 2011.
8. PRESS, Gil. *Internet of Things by the Numbers: Market Estimates and Forecasts*. 2014. Forbes.
9. DAVIS, Almir and CHANG, Hwa. A Survey of wireless sensor network architectures. *International Journal of Computer Science & Engineering Survey (IJCSES)*. 2012. Vol. 3, no. 6, p. 1–22.
10. DIGI INTERNATIONAL INC. Wireless Mesh Networking: ZigBee vs. Digi Mesh - White Paper. . 2008. P. 1–5.
11. KARL, Holger and WILLIG, Andreas. Single-Node Architecture. *Protocols and Architectures for Wireless Sensor Networks*. P. 15–57.
12. GRILO, António. Wireless Sensor Networks Chapter 2 : Single node architecture. .
13. FORTUNA, Carolina. Why is sensor data hard to get? . 2010.
14. CECÍLIO, José and FURTADO, Pedro. *Wireless Sensors in Heterogeneous Networked Systems*. Springer, 2014.
15. ABDELGAWAD, Ahmed and BAYOUMI, Magdy. *Resource-Aware data fusion algorithms for wireless sensor networks*. Springer Science & Business Media, 2012.

16. PAPER, White. Internet of Things : Wireless Sensor Networks Executive summary. .

17. ALLEMANG, Dean and HENDLER, James. *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier, 2011.

18. DENTLER, Kathrin, CORNET, Ronald, TEN TEIJE, A C M, DE KEIZER, Nicolette F and OTHERS. Comparison of reasoners for large ontologies in the OWL 2 EL profile. . 2011.

19. PARSIA, Bijan and SIRIN, Evren. Pellet: An owl dl reasoner. In : *Third International Semantic Web Conference-Poster*. 2004.

20. QUILITZ, Bastian and LESER, Ulf. *Querying distributed RDF data sources with SPARQL*. Springer, 2008.

21. KUMAR, Archana P, KUMAR, Abhishekh and KUMAR, Vipin N. A Comprehensive Comparative study of SPARQL and SQL. *International Journal of Computer Science and Information Technologies*. 2011. Vol. 2, no. 4, p. 1706–1710.

22. KNUBLAUCH, Holger, FERGERSON, Ray W, NOY, Natalya F and MUSEN, Mark A. The Prot{é}g{é} OWL plugin: An open development environment for semantic web applications. In : *The Semantic Web--ISWC 2004*. Springer, 2004. p. 229–243.

23. MCBRIDE, Brian. Jena: Implementing the RDF Model and Syntax Specification. In : *SemWeb*. 2001.

24. SHETH, Amit P. Ontological evaluation and validation. [online]. 2003. Available from: http://www.cs.uga.edu/~budak/papers/eval.pdf

25. GÓMEZ-PÉREZ, Asunción and EUZENAT, Jérôme. *The Semantic Web: Research and Applications: Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29--June 1, 2005, Proceedings*. Springer, 2005.

26. GUARINO, Nicola and WELTY, Christopher A. An overview of OntoClean. In : *Handbook on ontologies*. Springer, 2009. p. 201–220.

27. CROSS, Valerie and PAL, Anindita. OntoCAT: An ontology consumer analysis tool and its use on product services categorization standards. *CEUR Workshop Proceedings*. 2006. Vol. 226, p. 44–58.

28. LOZANO-TELLO, Adolfo and GÓMEZ-PÉREZ, Asunción. Ontometric: A method to choose the appropriate ontology. *Journal of Database Management*. 2004. Vol. 2, no. 15, p. 1–18.

29. LANTHALER, Markus and GÜTL, Christian. On using JSON-LD to create evolvable RESTful services. In : *Proceedings of the Third International Workshop on RESTful Design*. 2012. p. 25–32.

30. WAGH, Kishor and THOOL, Ravindra. A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host. *International Journal of Information Engineering and Applications*. 2012. Vol. 2, no. 5, p. 12–17.

31. SHELBY, Z, HARTKE, K and BORMANN, C. *RFC 7252: The Constrained Application Protocol (CoAP)*. 2014. ISBN 2070-1721.

32. PAPER, Technical White. Service Oriented Architecture (SOA) and Specialized Messaging Patterns. *Architecture* [online]. 2007. Vol. 345, p. 1–15. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.5751&rep=rep1&type=pdf

33. FERBER, Jacques. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley Reading, 1999.

34. ROBIN, Alexandre. OGC SWE Common Data Model Encoding Standard. *OGC 08--094r1*. 2011.

35. COX, S. *OGC Implementation Specification 07-022r1: Observations and Measurements-Part 1-Observation schema. Open Geospatial Consortium*. 2007.

36. BOTTS, Mike and ROBIN, Alexandre. OpenGIS sensor model language (SensorML) implementation specification. *OpenGIS Implementation Specification OGC*. 2007. Vol. 7, no. 000.

37. COMPTON, Michael, BARNAGHI, Payam, BERMUDEZ, Luis, GARCÍA-CASTRO, RaúL, CORCHO, Oscar, COX, Simon, GRAYBEAL, John, HAUSWIRTH, Manfred, HENSON, Cory, HERZOG, Arthur and OTHERS. The SSN ontology of the W3C semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*. 2012. Vol. 17, p. 25–32.

38. RUSSOMANNO, David J, KOTHARI, Cartik R and THOMAS, Omoju A. Building a Sensor Ontology: A Practical Approach Leveraging ISO and OGC Models. In : *IC-AI*. 2005. p. 637–643.

39. BENDADOUCHE, Rimel, ROUSSEY, Catherine, DE SOUSA, Gil, CHANET, Jean-Pierre and HOU, Kun Mean. Extension of the semantic sensor network ontology for wireless sensor networks: the stimulus-wsnnode-communication pattern. In : *5th International Workshop on Semantic Sensor Networks in conjunction with the 11th International Semantic Web Conference (ISWC)*. 2012. p. 16–p.

40. NIKOLIĆ, Siniša, PENCA, Valentin, SEGEDINAC, Milan and KONJOVIĆ, Zora. Semantic web based architecture for managing hardware heterogeneity in wireless sensor network. In : *Proceedings of the International Conference on Web Intelligence, Mining and Semantics*. 2011. p. 42.

41. EASTMAN, Roger, SCHLENOFF, Craig, BALAKIRSKY, Stephen and HONG, Tsai. A Sensor Ontology Literature. . 2013.

42. CANO, David Moner. CEN / TC251 EN13606 Electronic Health Record Communication About IBIME (I). .

43. MORI, Angelo Rossi and CONSORTI, Fabrizio. Exploiting the terminological approach from CEN/TC251 and GALEN to support semantic interoperability of healthcare record systems. *International journal of medical informatics*. 1998. Vol. 48, no. 1, p. 111–124.

44. LATUSKE, Rudi and GMBH, a R S Software. Bluetooth Health Device Profile and the IEEE 11073 Medical Device Frame Work Bluetooth in Medical Applications. *Work*. P. 1–6.

45. TECHNOLOGIES, Bluegiga. Bluetooth ® low energy technology. .

46. ASTM. ASTM International Technical Committee E31. [online]. 2007. P. 9555. Available from: http://www.astm.org/COMMITTEE/E31.htm

47. ISON, Jon, KALAŠ, Matúš, JONASSEN, Inge, BOLSER, Dan, ULUDAG, Mahmut, MCWILLIAM, Hamish, MALONE, James, LOPEZ, Rodrigo, PETTIFER, Steve and RICE, Peter. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. *Bioinformatics*. 2013. Vol. 29, no. 10, p. 1325–1332.

48. HODGSON, RALPH, KELLER, PAUL J, HODGES, JACK and SPIVAK, JACK. QUDT--Quantities, Units, Dimensions and Data Types Ontologies. *USA, Available from: http://qudt. org [March 2014]*. 2013.

49. ARORA, Kanika. International journal of Emerging Trends in Science and Technology. . 2014. P. 317–325.

50. TOWNSEND, By Erik. The 25 Year History of Service Oriented Architecture. . 2008.

51. MAHMOOD, Zaigham. Service Oriented Architecture: Potential Benefits and Challenges. *Proceedings of the 11th WSEAS International Conference on Computers*. 2007. P. 497–501.

52. MARTIN, David, BURSTEIN, Mark, MCDERMOTT, Drew, MCILRAITH, Sheila, PAOLUCCI, Massimo, SYCARA, Katia, MCGUINNESS, Deborah L, SIRIN, Evren and SRINIVASAN, Naveen. Bringing semantics to web services with OWL-S. *World Wide Web*. 2007. Vol. 10, no. 3, p. 243–277.

53. GUBBI, Jayavardhana, BUYYA, Rajkumar, MARUSIC, Slaven and PALANISWAMI, Marimuthu. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*. 2013. Vol. 29, no. 7, p. 1645–1660.

54. DE, Suparna, BARNAGHI, Payam, BAUER, Martin and MEISSNER, Stefan. Service modelling for the Internet of Things. In : *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*. 2011. p. 949–955.

55. NAMBI, S N, SARKAR, Chayan, PRASAD, R Venkatesha and RAHIM, Abdul. A unified semantic knowledge base for IoT. In : *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. 2014. p. 575–580.

56. WANG, Wei, DE, Suparna, TOENJES, Ralf, REETZ, Eike and MOESSNER, Klaus. A comprehensive ontology for knowledge representation in the internet of things. In : *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. 2012. p. 1793–1798.

57. HACHEM, Sara, TEIXEIRA, Thiago, HACHEM, Sara, TEIXEIRA, Thiago, HACHEM, Sara, TEIXEIRA, Thiago and ISSARNY, Valérie. Ontologies for the Internet of Things To cite this version : Ontologies for the Internet of Things. . 2011.

58. REPORT, Technical. Study on Semantic support for M2M Data. . 2013. Vol. 1, p. 1–33.

59. HACHEM, Sara, TEIXEIRA, Thiago and ISSARNY, Valérie. Ontologies for the Internet of Things. *ACMIFIPUSENIX 12th International Middleware Conference* [online]. 2011. No. June 2009, p. 3:1–3:6. DOI 10.1145/2093190.2093193. Available from: http://doi.acm.org/10.1145/2093190.2093193

60. TRANSPORT, MQTT M Q Telemetry. *V3. 1 Protocol Specification*. 2014.

61. BIZER, Christian, HEATH, T and BERNERS-LEE, T. Linked data-the story so far. *International journal on Semantic Web and Information Systems* [online]. 2009. Vol. 5, no. 3, p. 1–22. DOI 10.4018/jswis.2009081901. Available from: http://eprints.soton.ac.uk/271285/

62. STANDARD, The. What is HL7 ? . P. 1–3.

63. GRABENWEGER, J and DUFTSCHMID, G. Ontologies and their application in electronic health records. *eHealth2008 Medical Informatics meets eHealth, Wien*. 2008. P. 29–30.

64. BEALE, T, HEARD, S, LLOYD, D and KALRA, D. EHR Information Model. [online]. 2007. Available from: http://discovery.ucl.ac.uk/66193/

65. SWETINA, Jorg, LU, Guang, JACOBS, Patricia, ENNESSER, Francois and SONG, JAESEUNG. Toward a standardized common m2m service layer platform: Introduction to onem2m. *Wireless Communications, IEEE*. 2014. Vol. 21, no. 3, p. 20–26.

66. LAN, Shi, QILONG, Miao and DU, Jinglin. Architecture of wireless sensor networks for environmental monitoring. In : *Education Technology and Training, 2008. and 2008 International Workshop on Geoscience and Remote Sensing. ETT and GRS 2008. International Workshop on*. 2008. p. 579–582.

67. AVILÉS-LÓPEZ, Edgardo and GARCÍA-MACÍAS, J Antonio. TinySOA: a service-oriented architecture for wireless sensor networks. *Service Oriented Computing and Applications*. 2009. Vol. 3, no. 2, p. 99–108.

68. DELICATO, Flávia C, PIRES, Paulo F, PIRMEZ, Luci and BATISTA, Thais. Wireless sensor networks as a service. In : *Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on*. 2010. p. 410–417.

69. GHOBAKHLOU, Akbar, KMOCH, Alexander and SALLIS, Philip. Integration of Wireless Sensor Network and Web Services. In : *Proceedings of the 20th International Congress on Modelling and Simulation, Adelaide, Australia*. 2013.

70. KHEDO, Kavi Kumar and SUBRAMANIAN, R K. A Service-Oriented Component-Based Middleware Architecture for Wireless Sensor Networks. *Journal of Computer*

Science [online]. 2009. Vol. 9, no. 3, p. 174–182. DOI 10.1016/j.foreco.2007.12.007. Available from: http://paper.ijcsns.org/07_book/200903/20090324.pdf

71. SINGH, Akhilendra Pratap, VYAS, O P and VARMA, Shirshu. Flexible Service Oriented Network Architecture for Wireless Sensor Networks. *International Journal of Computers Communications & Control*. 2014. Vol. 9, no. 5, p. 610–622.

72. AMATO, Flora, CASOLA, Valentina, GAGLIONE, Andrea and MAZZEO, Antonino. A semantic enriched data model for sensor network interoperability. *Simulation Modelling Practice and Theory*. 2011. Vol. 19, no. 8, p. 1745–1757.

73. CHU, Xingchen. *Open sensor web architecture: core services*. The University of Melbourne, Australia, 2005.

74. BRANDT, Paul, BASTEN, Twan and STUIJK, Sander. ContoExam: an ontology on context-aware examinations. *Garbacz, P. Kutz, O., 267, 303-316*. 2014.

75. TEST, Continua and VERSION, Certification Plan. 1.4, 2009. *Continua Health Alliance*. 2009.

76. WARTENA, Frank, MUSKENS, Lohan, SCHMITT, Lars and PETKOVIC, M. Continua: The reference architecture of a personal telehealth ecosystem. In : *e-Health Networking Applications and Services (Healthcom), 2010 12th IEEE International Conference on*. 2010. p. 1–6.

77. BJERREGAARD, Jensen H and DUEDAL, Pedersen C. MedCom: Danish health care network. *Studies in health technology and informatics*. 2003. Vol. 100, p. 59–65.