University of Piraeus
Department of Digital Systems
Postgraduate Program “Digital Communications and Networks”

Enhanced management of IoT devices based on collective knowledge

Supervisor: Dimosthenis Kyriazis

Christos Pentsioudis

Contents

Πανεπιστήμιο Πειραιώς

Πανεπιστήμιο Πειραιώς

Abstract

The purpose of this project is to develop an environment where different devices from our daily life (e.g. smart phone, digital camera, our car etc.) will communicate to each other and will exchange information and data and act according the data and information acquired. To achieve this we used as a base the idea of Internet of Things (IoT). In the Internet of Things every device is connected and will exchange data and information. Exchanging those information will make our daily life easier and will help us in many different ways to achieve numerous things that today is impossible to do. According to abovementioned we developed an Android application that enables sharing of experiences and allows IoT devices to act according to these shared experiences received. The developed application is a representative example of the added value brought with respect to distributed coordination and autonomous management based on the knowledge that can be shared amongst communities of IoT devices.

Πανεπιστήμιο Πειραιώς

Περίληψη

Ο σκοπός της παρούσας εργασίας είναι να αναπτύξει ένα περιβάλλον όπου διαφορετικές συσκευές από την καθημερινή μας ζωή (π.χ. έξυπνο κινητό τηλέφωνο, ψηφιακή φωτογραφική μηχανή, το αυτοκίνητο μας κλπ.) θα επικοινωνούν μεταξύ τους ώστε να είναι εφικτό να ανταλλάσσουν πληροφορίες και δεδομένα και να δρουν ανάλογα με τα δεδομένα και τις πληροφορίες που αποκτήθηκαν. Για να επιτευχθεί αυτό χρησιμοποιήσαμε σαν βάση μας την ιδέα του Internet of Things (IoT). Στο Internet of Things (IoT) κάθε συσκευή θα είναι συνδεδεμένη και θα ανταλλάσσει πληροφορίες με κάποια άλλη. Οι ανταλλασσόμενες αυτές πληροφορίες θα κάνουν την ζωή μας πιο εύκολη και θα μας βοηθήσει σε πολλά πράγματα που σήμερα είναι αδύνατο να επιτευχθούν. Σύμφωνα με τα παραπάνω έχουμε αναπτύξει μια εφαρμογή Android που επιτρέπει την ανταλλαγή εμπειριών και επιτρέπει στις συσκευές IoT να ενεργούν σύμφωνα με αυτές τις κοινές εμπειρίες που έλαβε. Η εφαρμογή που αναπτύχθηκε είναι ένα αντιπροσωπευτικό παράδειγμα της προστιθέμενη αξία που έφερε σε σχέση με το κατανεμημένο καταμερισμό (distributed coordination) και την αυτόνομη διαχείριση με βάση τη γνώση που μπορεί να μοιραστεί ανάμεσα στις κοινότητες των συσκευών IoT.

Πανεπιστήμιο Πειραιώς

1. Introduction

Over 2 billion mobile terminals that are in use today and over 1 billion users that use the Internet worldwide. Wireless, mobile and web technologies gave the ability to communicate between systems and applications. However, this is only the start as an example the radio frequency identification (RFID tags).

Furthermore, is expected simple tags to involve into smart things connected to a network with increased storage, processes also the ability to add sensors. This will open new possibilities to network applications such as Man-to-machine, Machine-to-mobile and Mobile-to-machine and context-aware communications.

The Internet of Things (IoT) [1] is a network from billion or trillion of machines which are communicating between them. This is important for the development of the information and the communication, because a simple example of those are already is use today. As we speak there are 1.3 billion RFIDs and 2 billion users of mobile services in the whole world.

The current thesis presents an approach to enable autonomous management of IoT objects based on the information (i.e. experience) received from other objects. Based on an extended ontology, a cloud-based mechanism allows objects to share the aforementioned information and act accordingly.

1.1 Structure

The chapter 1 introduces the related technology of IoT, the architecture and what the prospects are. In the next chapter we analyze the problem statement and some solutions. The third chapter is dealing with the state of the art ontologies that already exists. In the fourth chapter we will analyze the ontology that will be used in order to solve the problem. The fifth chapter is dedicated to the analysis of the developed application. Finally we will have the conclusion and some future work about the application we've developed.

1.2 Internet of Things

1.2.1 Overview

Several architectural paradigms have been proposed. One of them is the IoT-A[2], which is following a spiral design and development model. In order to describe this architecture we'll execute a modeling exercise. This exercise will help us understand better the IoT domain. Also, this exercise allows the repeats the process of building the architecture, so the stability of the models as part of the architecture is increased. Each repeat generates additional updates and content for this architecture description, as the understanding of the application IoT domain increases. An architecture methodology is defined to ensure consistency of the

architecture description during each iteration. The architecture model is mostly describes dependencies between models (i.e., the IoT Reference Model guides the definition of the IoT Reference Architecture).Once a change is proposed in one modelling aspect, a clear chain of dependencies can be followed. In this way, the overall consistency of the IoT-A reference model architecture is achieved.

Reference model and reference architecture

The reference models and architectures providing a description of greater abstraction than is inherent to actual systems and applications. They are both more abstract than the system architectures that have been designed for a particular application. From the literature, we can expend the dependencies of reference architecture, architectures, and actual systems. Architecture is helping in designing, engineering, building, and testing actual systems. However, have to understand the systems constraints better so we can provide input to the architecture design, and in turn identifying future opportunities. The structure of the architecture can be made explicit through an architecture description, or it is implicit through the system itself. In order to define an architecture reference, we have to extract the essentials of existing architectures, like mechanism or usage can be defined. Areference architecture can provide guidance in form of best practices. Such guidance can, for instance, make new architectures and systems compliant to each other. These general architecture dependencies apply to the modeling of the IoT domain as well.

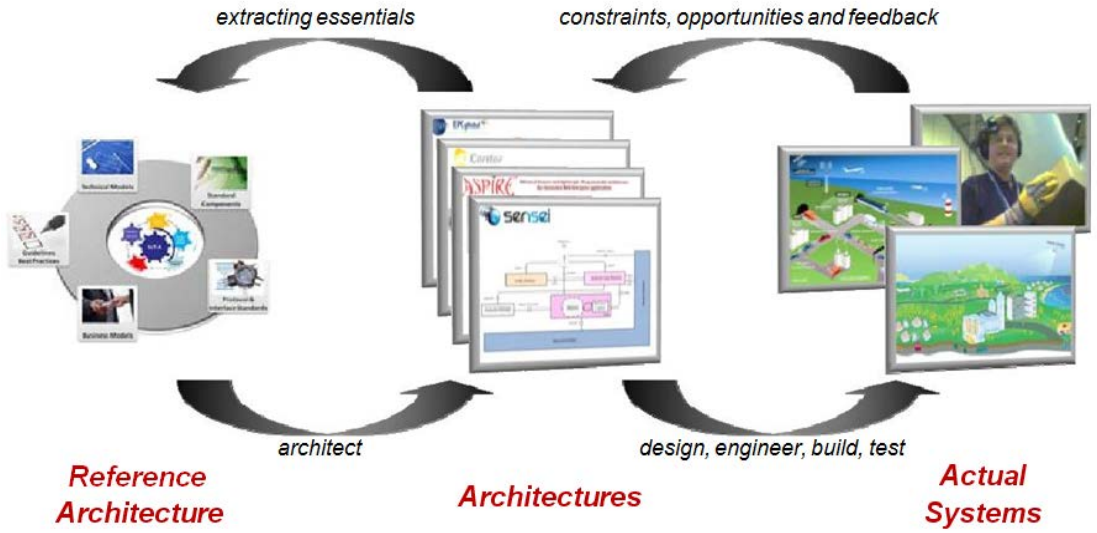


Figure 1. Relationship between a reference architecture, architectures, and actual systems (adapted from Mueller)[3]

While the model presented in the picture above stops at the reference architecture, the IoT-A architecture model continues a step further and also define a reference model. As we already mentioned earlier, the reference model is providing us a

common understanding of the IoT domain, this is achieved by modeling its concepts and their relationships.

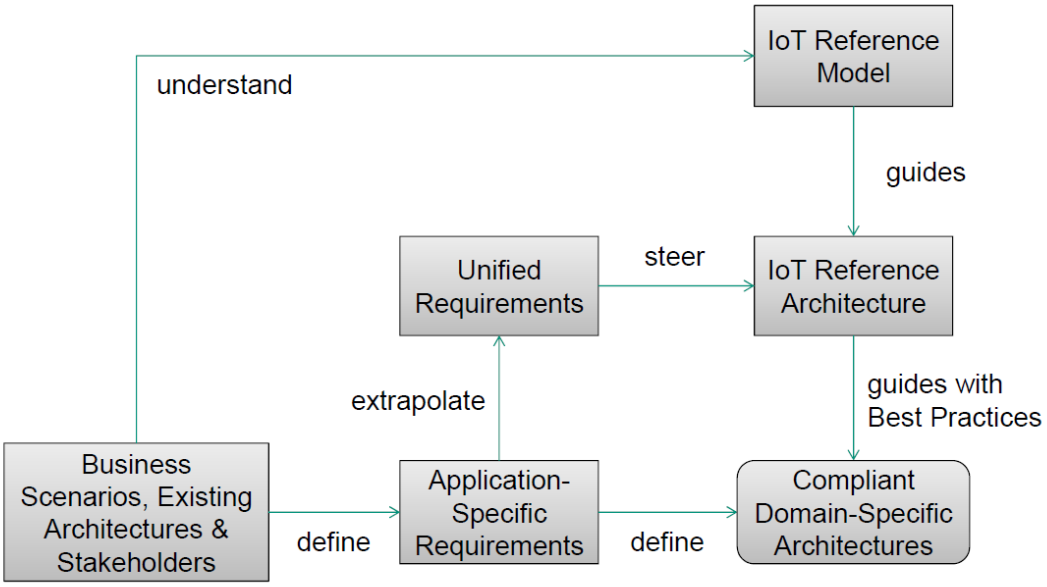


Figure 2. High-level taxonomy of the IoT reference model and IoT reference architecture dependencies and model influences[4]

In the picture above, the inputs and dependencies of the IoT reference model and the IoT reference architecture are depicted.

The IoT Reference Model provides guidance for the description of the IoT reference architecture. The IoT reference architecture in turn guides the definition of compliant domain specific architectures. Essential inputs for the definition of the IoT reference model are stakeholder concerns, business scenarios, and existing architectures. Important here is to create a common understanding of the IoT domain from the different inputs. This is mainly a modeling exercise, during which experts have to work together and extract the main concepts and their relations of the IoT domain from available knowledge. Furthermore, business scenarios, existing architectures, and stakeholder concerns can be transformed into application specific requirements. When extrapolated, these requirements lead to a set of unified requirements. Unified requirements in turn steer the definition of the IoT Reference Architecture. A more detailed explanation of what guides the IoT-A architecture-modeling process is provided in the next Section.

1.2.2 Infrastructure Technologies

IPv6

From the start of the internet we used the IPv4 Internet protocol. However due to development of technology more and more IPv4 has been required, So the IPv4 it's

almost exhausted. The Internet of Things need a lot more IPs this sold by using the new Internet by using the protocol IPv6[6].This protocol expands the current Internet Protocol (IPv4). The IPv6 offers better addressing, security and more features to support large worldwide network, that make it perfect for Internet of Things (IoT). The IPv6 working with a new 128-bit system compared with the IPv4 that it used 32-bit system. For example an IPv4 would be like this “192.168.100.32” at IPv6 would be like this “0000:0000:0000:0000:0000:0000:C0A8:6420” or “::C0A8:6420”. So the IPv6 have been designed with scalability and extensibility, this will allow the Internet Of Thing to develop without the addressing problem.

RFID System

One of the most popular technologies of identification that is recommended by the ITU is the Radio Frequency Identification (RFID)[6]. RFID is an automatic method of identification that is using radio waves and it can locate elements for example human, animal or even things. RFID is the new barcode, but instead of bars it’s using a microchip with radio waves. Those labels are wide known as electronic label, and there are consisting by a RFID chip and an antenna. Those labels can be read a lot of meter, the visual contact is unnecessary. The most common method to read something like this is to increasing number that is determined or by a human or by another thing, but one RFID can save data and other information. RFID system use those labels, those labels consists the most popular and efficient technology for the identification system of IoT.

One RFID label can transfer an ID that can be a unique string or a password. It’s a special device that is designed to send a determined response to a specific signal. There different states of RFIDs, so there as passive, semi-passive and active. Those states can be determined by their power source, also a RFID can only read, read/write and read and write. Also there furthermore two types of RFID: the near field RFID and the distance field RFID. The near field RFID is the wide known NFC (Near Field Communication) is widely used from smart phone and to credits cards too. For example you can assign your credit card to your smartphone and pay your bill, transfer your money simple from your phone. This type of chip can be activated from the distance of 5 cm or by touch. The distance field RFID cover distances between 3 to 6 meters and it’s usually used by stationary reader.

Wireless Sensor Networks

The technological progress helped us to develop new low power integrated circuits, and also the wireless communications have been more available and efficient, low

cost and power devices can be used for remotely sensing applications. Those factors improved the viability of utilizing a sensor network the will be contained a large number of sensors, enabling the collection, processing, analysis and dissemination of valuable information, gathered from various environments. The RFID that there currently in use are the as the lower end Wireless Sensor Networks (WSN)[7]nodes but there have limited processing capabilities and storage. We have to overcome those scientific challenges in order to realize the full potentials of the WSNs that are substantial and multidisciplinary. The sensors data are being shared among the sensors nodes and sent to a distributed centralized system for analysis. The parts that compose the WSN monitoring network are:

- WSN hardware

One WSN core hardware contains sensor interfaces, processing units, transceiver units and power supply. Almost always, they comprise of multiple A/D converters for sensor interfacing and more modern sensor nodes have the ability to communicate using one frequency band making them more versatile.

- WSN communication stack

The nodes of this technology are deployed in an adhoc protocol for the most applications. In order to ensure the scalability and longevity of this network it's necessary to design a proper topology, routing and MAC layer. Also the nodes of the WSN can communicate among each other so it can make possible to transmit data in a single or in a multi- hop to a base station. But also the node drop out, and consequent degraded network lifetimes, are frequent. Finally this communication stack it will interact whit the outside world through the Internet and it can act like a gateway to the WSN subnet and the Internet.

- Middleware

The middleware mechanism is responsible to combine the cyber infrastructure with a Service Oriented Architecture (SOA) and sensor network so it can provide access to heterogeneous sensor resources for deployment independent manner. Those can be used by several applications. But in order to work this platform is required to develop sensor applications, such application is an Open Sensor Web Architecture (OSWA). This application is built according to certain set of operations and standard data representations as defined in the Sensor Web Enablement (SWE) by the Open Geospatial Consortium (OGC).

- Secure Data aggregation

An efficient and secure data aggregation method is required so we can extend the lifetime of the network and to ensure that reliable data has been collected from the sensors. A common problem of the WSN is a node failure, the network topology it should have the capability to heal itself. It is critical to ensure the security as the system is automatically connected to actuators and the protection of the system from intruders is a very important cause.

1.2.3 Application domains

Some examples of the usage of Internet of Things are the following:

- The Health

In the next decades, the model of healthcare will transform from the present hospital-centric to hospital-home-balanced in 2020th.And from hospital-home-balanced to the final home-centric in 2030th. The healthcare system will follow the layer structure for example from low to high comprising the personal, home, community and the hospital layer. Also the lower layer has lower labor and also lower operational cost, it's used frequently from people with chronic disease. So this type of healthcare at home service enabled by the IoT technology and is promising traditional healthcare industry and the ICT industry. The Health-IoT service is ubiquitous and personalized and will speed up the transformation of healthcare from career-centric to patient-centric.

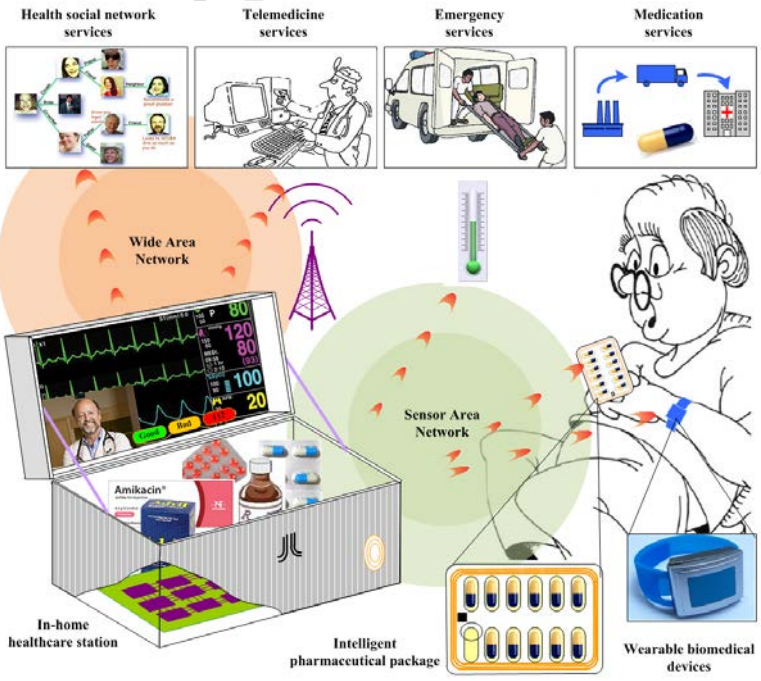


Figure 3. Senario for in-Home healthcare solution [8]

Typically, a Health-IoT solution includes the following functions:

1. Tracking and monitoring. Powered by the ubiquitous identification, sensing, and communication capacity, all the objects (people, equipment, medicine, etc.) can be tracked and monitored by wearable WSN devices on a 24/7 basis.

2. Remote service. Healthcare and assist living services e.g. emergency detection and first aid, stroke habitation and training, dietary and medication management, telemedicine and remote diagnosis, health social networking etc. can be delivered remotely through the internet and field devices.

3. Information management. Enabled by the global connectivity of the IoT, all the healthcare information (logistics, diagnosis, therapy, recovery, medication, management, finance, and even daily activity) can be collected, managed, and utilized throughout the entire value chain.

4. Cross-organization integration. The hospital information systems (HISs).are extended to patient' home, and can be integrated into larger scale healthcare

- Smart Cities

There will be all kind of sensors to measure a lot of variables so to achieve a better town for the citizens.

- Noise Sensors

Noise sensors will help us measure the noise over the city, for example in the regions with bars or clubs and the music is too loud an automatic alarm will inform the police.

- Electromagnetic Sensors

One smart city also could have electromagnetic sensors to measure and to control the radiation level from Wi-Fi and from cell stations so it can help the heath of the citizens.

- Smart Lighting

There will be sensors one everyone light on the streets, with those sensors the power consumption will be reduced significantly. Also the lights will adapt to the weather of the city for example if today the weather is too cloudy the lights will remain on. Also at night in the neighborhoods there are streets that too little citizens are circulating so the lights will turn off and if there a move detected will back turn on.

Πανεπιστήμιο Πειραιώς

- Smart Parking

Mostly in all large cities around the world the drivers are searching for parking for a long time even hours. In a smart city there will be sensors at the parking spots. Those sensors will measure is a parking spot is empty or not, also will cooperate for example with an Android application. If a driver is searching for a parking will send a request from the application and if a parking lot is free nearby the application will alert him. The Smart Parking is actually working nowadays at Boston City.

- Smart factory

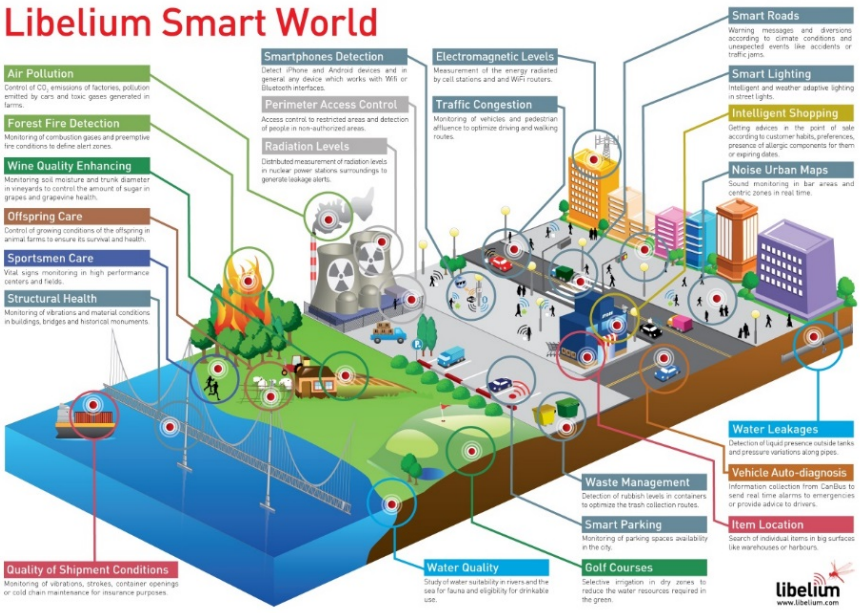


Figure 4. The future Smart City [9]

The companies will be able to track all their products by means of RFID tags in a global supply chain; as a consequence, companies will reduce their operational expenditure and improve their productivity due to a tighter integration with enterprise resource planning and other systems. The IoT will provide automatic procedures that will apply a reduction in the number of the employees needed. The workers will be replaced with bar-code scanners, readers, sensors and actuators. Without any doubt, these technologies will bring opportunities for workers and a big number of technicians will be necessary needed to program and repairs all those machines.

- Thermostats, Fire alarm and CO alarm

This smart thermostat is connected to Internet and can be regulated from distance. This device learns your preferences about the temperature, and after a while will learn your habits for example your daily program. When you are away it can be turned -off automatically, so to reduce the consumption of energy and save money. Also you can program it so your house have the perfect temperature the time you wanted for example you want at 6 a.m. to have 26 0C, to do this it will start automatically earlier since the device knows how much time take to heat up your home. However you can control it from your phone and you can change the settings any time you even when you are miles away from your home. Also the device can export monthly reports of the consumption, the energy it has been saved. The fire alarm will inform you early enough for the possibility of a fire in your home and the CO levels at any time with an alarm sound and with a light indicator. It'll be the ability to send you a notice at your smartphone in case of an emergency.

- Smart monitor for babies

This device would monitor your baby, with a non-contact sensor. This sensor would monitor the temperature, the heart rate of the baby. With this device also you can listen to your baby and know when is sleeping. The best of this monitor is that the parent is able to see all those information live at his smart phone. The application on the smart phone can notice you also when you have to feed the baby or even it can remind you the schedule dose of a medicine.

- Forest fire detection

There will be sensors all around the forest. If a fire starts those sensors will immediately notice the fire department and also start the extinguishing system. This system will help us to reduce the response time of the fire department that is too important in such cases.

- Smart Water

There will sensors that can measure the water quality, the water leakages and river floods. With the water quality sensors will be able study the suitability of the drinking water in almost every house. With the water leakage will help us with the Detection of liquid presence outside tanks and pressure variations along pipes. And the river floods is important for cities with big rivers near them, will monitor the level of the water, dams and reservoirs. With this system can prevent floods before even we know is going to happen.

Πανεπιστήμιο Πειραιώς

- Environment

Applications in the environmental domain have many overlaps with other scenarios. All those application will help us save energy. One prominent example is Smart Grid. Concerning this application area one needs to highlight initiatives that imply a more distributed energy production, since many houses have a solar panel today. As a vital part, smart metering is considered as a pre-condition for enabling intelligent monitoring, control, and communication in grid applications.

The use of IoT platforms in Smart Metering will provide the following benefits:

- An efficient network of smart meters allows for faster outage detection and restoration of service. Such capabilities redound to the benefit of customers.
- Provides customers with greater control over their energy or water consumption, providing them more choices for managing their bills.
- IoT deployment of smart meters is expected to reduce the need to build power plants. Building power plants that are necessary only for occasional peak demand is very expensive. A more economical approach is to shape the demand by either to incentivize customers to reduce their demand through time-based rates or other programs, or by service-level agreements that allow temporarily turning off devices which are not needed (e.g., the freezer for 20 minutes).

In order to describe a well-defined business model it is necessary to define what needs to be done in the business, which are the metrics for success, which are the problems that must be solved and the plans that solve these problems. Knowing which part of the problem is possible to solve and how much time is needed and which part cannot be solved is an important step that we must take into account when we develop concrete business cases for some of the application fields discussed above. As we can only go into the details of one business case in the context of this document, we will pick a use case from the application field of retail, as this is a central application field for the project, and apply an appropriate business case methodology to it. This methodology is outlined in the next section. The use of a methodology instead of merely calculating “some kind of business case” enables us to perform comparisons between different application fields, for instance when we consider health under an economic perspective within the context of the forthcoming deliverables.

2. Problem Statement

2.1 Context

The main idea is to enable different objects to acquire social skills and make individuals and communities to act as objects. Current approaches are focused on centralized management mechanisms, but taking into consideration that the number of devices and assets is increasing (e.g. sensors in a smart city environment, data objects in data centers, etc.), the aim of future IoT is to enable more decentralized and autonomous management for these devices and assets. The latter is of major importance given that objects may also belong to different administrative domains and thus centralized management information and decisions may not be feasible. The objects are the basic thing of an IoT network and in order to optimize their operation and their management mechanisms are required to minimize the amount of management information and the management components in IoT communities and allow them to react in a more autonomous way. The latter will address the following challenges:

- Objects interact and exchange information. The challenge and goal is to enable objects to exchange information and utilized it to drive management decisions.
- Protocols are required to facilitate information exchange. Given that different kinds of information can be exchanged (from objects with different characteristics, different APIs, etc), the challenge is to develop a common "language" and way (as a protocol) that will enable objects to report experiences and current statue.
- Mechanisms that will allow objects to learn from others. Like humans, objects experience various situations (e.g. the memory required for executing an application). The challenge is to allow objects to learn from others and thus act in a more autonomous way by exploiting this knowledge.
- Adapt to different situations. Every object on such a network / community will provide an experience (as feedback), which can be used by other things (e.g. resolution or zoom settings of a camera) so they could adapt to different situations.
- Becoming more intelligent. Sharing and gaining information from objects, will help to reinforce service delivery through more intelligent objects, considering both functional parameters such as performance, availability, reliability etc. and non-functional such as for example trust, users experience, expectations etc.

Πανεπιστήμιο Πειραιώς

2.2 Proposed approach

There are a lot ways in which we can collect data and analysis with purpose to identify the relationships between different entities also we can find events that influence the behaviour of the things. In an exemplar scenario of a data center that uses data objects to store and retrieve data, harvesting and analysis of raw data will enable identification of relationship patters between data objects (e.g. retrieved in sequel for a specific application), detection of events (e.g. triggering live-migration), identification of data objects of high importance (e.g. that may lead to replication of these), etc.

The goal of the proposed approach is to utilize the generated data in order to enable things to act in a more autonomous way. The latter can be achieved through mechanisms that allow objects to exchange information and use it if they experience the same situation. Thus, the things will receive this information that will permit them to operate autonomous according to the new information that they will receive (e.g. new data objects created in the data center or increasing number of requests for data from a specific location). All the above requires new metadata structures that will include the information being shared by other things. These metadata structures – ontologies - will capture both the main attributes (e.g. lifecycle properties, access properties, quality of service properties, etc.) and the attributes concerning the appliance of social media technologies (e.g. relationships with other entities, importance, experiences of others, triggering conditions, etc.).

What is more, tools are required that will allow objects to share their experiences and exploit the experiences of other objects. Such tools are the first step towards autonomous reasoning of things given that the use of experiences will minimize the need to obtain management information from centralized mechanisms, while network overheads will also be minimized.

Πανεπιστήμιο Πειραιώς

3. State of the Art

3.1 Ontologies

3.1.1 SensorML

SensorML is a generic data model in UML for capturing classes and associations that are common to all sensors. This ontology is a part of an Open Geospatial Consortium (OGC)[10] initiative to contribute to the development of a Sensor Web. Snapshot of classes and associations provided by SensorML[11] can be used to create specific profiles, which facilitate the processing, geolocation and integration of observed data from millions of sensors. The profiles of individual sensors the use and/or extend SensorML concepts can be created and posted to the Web environment in which they can task, queried by monitoring and processing systems.

SensorML has been developed as a specification for efficient implementation by vendors that desire to implement OGCcompliant sensor system, therefore, SensorML try to specify as few class and relation definitions as possible. However, the ontological engineering focuses on rich semantic data and knowledge models. Therefore the re-conceptualization, redefinition, or extension of some of the classes in SensorML is necessary. SensorML has been developed by using syntax from the eXtensible Markup Language (XML). The use of XML syntax alone to define classes restricts the potential scope of interoperability and reuse of its sensor profile instantiations. SensorML does not include formal definitions of the classes or relations it uses, that is, it provides no logical or axiomatic-grounded theory to account for its conceptualizations and therefore cannot be considered to be ontology. However, SensorML is providing a generic data model that expresses the knowledge and data about sensors and can provide good schema for developing a data store for sensor metadata and sensed attribute values. Moreover, SensorML provides good organizational framework within the sensor ontology can be defined.

The essential elements to work the SensorML are:

- Component
  Is a physical atomic process that transforms information from one form to another.
- System
  Composite physically based model of a group or array of components, which can include detectors, actuators, or sub-systems. A System relates a Process Chain to the real world and therefore provides additional definitions regarding relative positions of its components and communication interfaces.
- Process Model
  Atomic non-physical processing block usually used within a more complex Process Chain. It is associated to a Process Method which defines the process

interface as well as how to execute the model. It also precisely defines its own inputs, outputs and parameters.

- Process Chain

  Composite non-physical processing block consisting of interconnected sub-processes, which can in turn be Process Models or Process Chains. A process chain also includes possible data sources as well as connections that explicitly link input and output signals of sub-processes together. It also precisely defines its own inputs, outputs and parameters.

- Process Method

  Definition of the behavior and interface of a Process Model. It can be stored in a library so that it can be reused by different Process Model instances (by using 'xlink' mechanism). It essentially describes the process interface and algorithm, and can point the user to existing implementations.

- Detector

  Atomic component of a composite Measurement System defining sampling and response characteristic of a simple detection device. A detector has only one input and one output, both being scalar quantities. More complex Sensors such as a frame camera which are composed of multiple detectors can be described as a detector group or array using a System or Sensor. In SensorML a detector is a particular type of Process Model.

- Sensor

  Specific type of System representing a complete Sensor. This could be for example a complete airborne scanner which includes several Detectors (one for each band).

The advantages of SensorML are:

- Electronic Specification Sheet

  SensorML is providing a standard digital means of providing specification sheets for sensor components and systems.

- Discovery of sensor, sensor systems, and processes

  The sensors system or processes can make themselves known and discoverable, so SensorML can detect them. Also is providing a rich collection of metadata that can be used to discover sensor system and observation processes. This metadata could include identifiers, classifiers, constraints, capabilities, characteristics, contacts, and references, in addition to inputs, outputs, parameters, and system location.

- Lineage of Observations

  In other words, it can describe in detail the process by which an observation came to be from acquisition by one or more detectors to processing and perhaps even interpretation by an analyst. Not only can this provide a

confidence level with regard to an observation, in most cases, part or all of the process could be repeated, perhaps with some modifications to the process or by simulating the observation with a known signature source.

- On-demand processing of Observations
  Process chains for geolocation or higher-level processing of observations can be described in SensorML, discovered and distributed over the web, and executed on-demand without a prior knowledge of the sensor or processor characteristics. This was the original driver for SensorML, as a means of countering the proliferation of disparate, stovepipe systems for processing sensor data within various sensor communities. SensorML also enables the distribution of processing to any point within the sensor chain, from sensor to data center to the individual user's PDA. SensorML enables this processing without the need for sensor-specific software.
- Support for tasking, observation, and alert services
  SensorML descriptions of sensor systems or simulations can be mined in support of establishing OGC Sensor Observation Services (SOS), Sensor Planning Services (SPS), and Sensor Alert Services (SAS). SensorML defines and builds on common data definitions that are used throughout the OGC Sensor Web Enablement (SWE) framework.
- Plug-N-Play, auto-configuring, and autonomous sensor networks
  This ontology had enabled the development of plug-n-play sensors, processes and simulations, which may be seamlessly added to Decision Support systems. Also the self-describing characteristic of this ontology had enabled sensors and processes also supports the development of auto-configuring sensor network and autonomous sensor networks, those sensors can publish alerts and task to which other sensor can subscribe and alert.
- Archiving of Sensor Parameters
  Finally, this ontology is providing mechanisms for achieving fundamental parameters and assumptions for sensors and processes, so those observations of those systems can also be reprocessed and be improved after their missions has ended. This characteristic is important for long-range applications like a global change monitoring and modeling.

3.1.2 O&M – OWL (SemSOS)

OWL[12] can be used when we want to process information that are included in documents, on the contrary to situations that the content only need to presented to humans. This ontology also can be used to represent the meaning of terms in vocabularies and moreover the relationships between those terms. So this exactly representation of terms and their interrelationships is called ontology. OWL has more characteristics for expressing meanings and semantics than XML, RDF[13] and

RDF-s, and furthermore OWL goes beyond these languages in its ability to represent machine interpretable content on the Web. OWL is a revision of the DAML+OIL web ontology language incorporating lessons learned from the design and application of DAML+OIL[15].

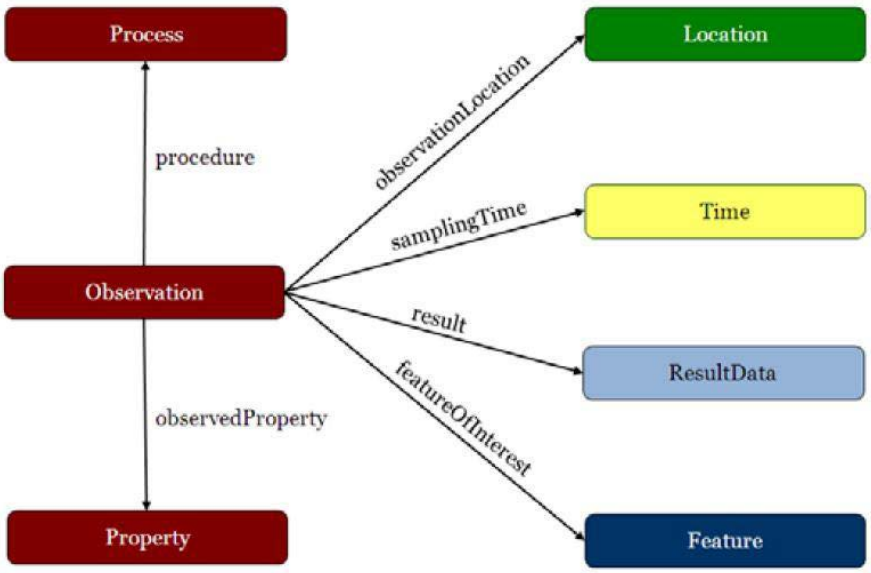


Figure 5. Subset of major concepts and relations in O&M-OWL [16]

The Semantic Web is a vision for the future of the Web. In this web the information will be given explicit, and will make easier for the machines to automatically process and integrate the information that are available at the Web. Also on Semantic Web we exploit the XML's ability to define and customize tagging schemes and RDF's flexible approach to represent data. The first level that is required above the RDF for the Semantic Web is an ontology language, this can describe the meaning of terminology that had been used in Web documents. The OWL Use Cases and Requirements have to go beyond the basic semantics of RDF schema and have to provide more details on ontologies. This is motivating the need for a Web Ontology Language in terms of six use cases, and formulates design goals, requirements and objectives for OWL.

- OWL has been designed to meet this need for a Web Ontology Language. OWL is part of the growing stack of W3C recommendations related to the Semantic Web.
- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with data types.

- RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and these data models can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties and characteristics of properties (e.g. symmetry), and enumerated classes.

3.1.3 Sleep Domain Ontology

The Sleep Domain Ontology (SDO) [17] purpose is to search relevant sensor data over distributed and heterogeneous sensor networks. However it's not available for use but it has been descripted in only two papers. There are some key frameworks such as:

i. Reuses the SUMO[18]ontology as a whole. The rest of the modules reference to the SUMO ontology.
ii. SensorHierarchyOntology.
    - Only a figure is available.
    - Describes Sensors and Sensor data.
    - Builtfrom IEEE 1451.4
        - Does not completely cover the standard
        - Some modelling errors: Mix subclass properties with ad-hoc properties (e.g., has, characterised_by)
        - Non-justified modelling decisions. E.g., they say that "A sensor is an actuator or a transducer" and in IEEE it appears that "A transducer is a sensor or an actuator".
iii. Sensor Data Ontology. There is nothing available. Fromthepaper:
    - "Describes the dynamic and observational properties of transducers data that goes beyond describing individual transducers"
    - "Describes the context of a sensor with respect to spatial and/or temporal observations"
    - "Provides abstract measurements/operations by groping transducers (virtual transducer)"
iv. Extension Plug-in Ontologies. There is nothing available. Each plug-in ontology includes the representation for a particular domain of sensor data and networks.

There no best feature but only a general concept such as:

i. Alignmentwithstandard.
ii. Alignmentwithupperontologies.
iii. Abstraction of groups of sensors/sensor networks in virtual sensors.
iv. Modulardevelopment.

As a weakest point refers that there's a lack information availability and some of them conflicts with standards.

3.1.4 Semantic Sensor Network Ontology

The Semantic Sensor Network (SSN) [19] has been developed by W3C Semantic Sensor Network Incubator Group (SSN-XG) so it can be describe sensors and observations, and other related concepts.

The SSN-XG group had two objectives:

i. To develop an ontology that can describe a sensor or an entirely network of sensors and those descriptions can be used in a sensor network or to a web application.
ii. Study and recommend methods for using the ontology to semantically enable applications developed according to available standards such as the OGC SWE standards.

After a review of the existing sensor and observation ontologies this ontology created. To do that the ontology for a starting point is using CSIRO Sensor ontology. SSN ontology defines a domain-independent and end-to-end model for sensing applications, by merging sensor-focused (e.g. SensorML), observation-focused (e.g. O&M) and system-focused views.

This ontology doesn't have any hierarchy of sensors types. These type of definitions left for the domain experts.

The ontology although is organized, conceptually (not physically) into ten modules. Those modules contains the classes and the properties that can be used to built aspects of a sensor or of its observations. The ontology can be used for focus on any of a number of perspectives such as:

i. sensor perspective, with a focus on what senses, how it senses, and what is sensed.
ii. data or observation perspective, with a focus on observations and related metadata.
iii. system perspective, with a focus on systems of sensors.
iv. feature and property perspective, with a focus on features, properties of them, and what can sense those properties.

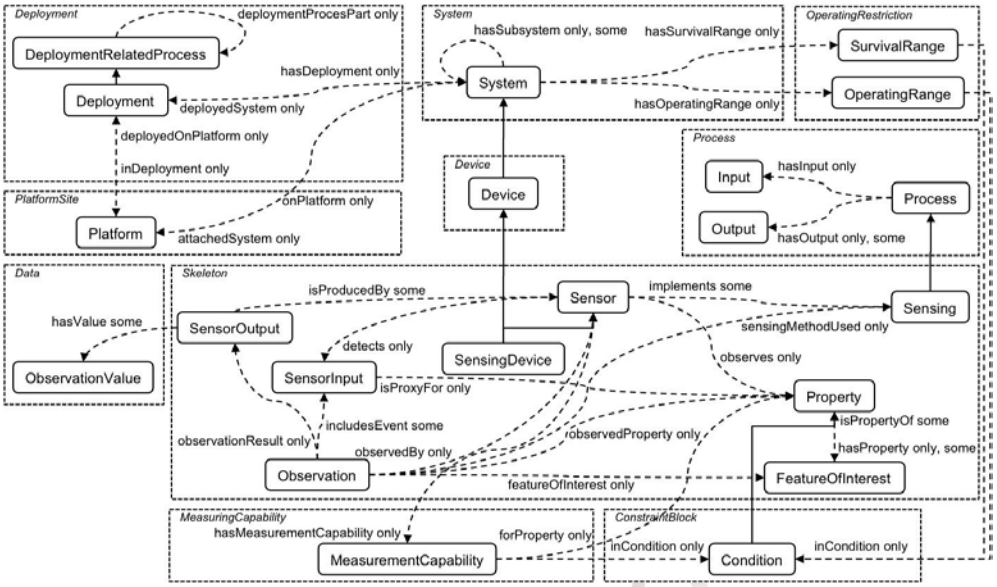


Figure 6. The SSN ontology, key concepts and relations, split by conceptual modules. The concepts not depicted are largely properties for measurement capabilities, and survival or operating ranges: accuracy, precision, resolution and the like. Note the central importance of sensors, observations and properties, brought out by the SSO ontology design pattern [20]

Finally close to the ontology Stimulus-Sensor-Observation ontology design pattern. This pattern links the sensors and the resulting observations:

- Observations act as the nexus between incoming stimuli, the sensor, and the output of the sensor, i.e., a symbol representing a region in a dimensional space. Properties are qualities that can be observed via stimuli by a certain type of sensors.
- Features of Interest are entities in the real world that are the target of sensing.
- Procedure is a description of how a sensor works, i.e., how a certain type of stimuli is transformed to a digital representation, perhaps a description of the scientific method behind the sensor.
- Result (or SensorOutput) is a symbol representing a value as outcome of the observation.

This pattern encompasses three of the four perspectives — the missing system perspective is more about system organization and deployments than sensing, but clearly links to the pattern.

3.1.5 CSIRO (Commonwealth Scientific and Industrial Research Organization) Sensor Ontology

The CSIRO [21] ontology created for describing and reasoning about sensors observations and scientific models. Semantic description of sensors have been provided by the use in workflows. This ontology have been developed but it's not completed or finished because it should provide a language to specify sensors but is agnostic about domain sensors, units of measurement, location etc. The observation and the sensors has an equivocal concept , for example a sensor ontology could be a sensor and observation ontology but hasn't happened yet.

The best features of this Ontology is:

i. Plug and Play – this ontology focus on the sensor and not in the domain concerns such as measurements, locations, etc.

ii. Hierarchy of Sensors – There is no hierarchy in this ontology in other words a thermometer is only a temperature sensor. Approaching the hierarchy of sensors there are a) more domain approach influence and b) can origin from another sensor.

iii. Composition – It is considered that this ontology is the only ontology which can do a proper composition.

3.1.6 OntoSensor

This ontology was created by the University of Memphis and purpose is to build a knowledge base of sensors. This base can be searched by the Protégé plugin. It is either active or complete and it hasn't been updated since 2008. Mainly is used to cover the sensor which listed in Crossbow catalogue. OntoSensor[22] contains a hierarchy of sensor classes and describes sensor attributes, services and capabilities. Also is focused on technical specification of sensors such as data acquisition boards, sensing elements and processor/radio units in the Crossbow 2006 catalogue. The sensors are sorted in minimum range, but several complicated properties are included.

This version is not autonomous it depends on other upper ontologies some of those are:

i. IEEE SUMO: Process, ContentDevelopment, MeasuringDevice, TransportationService
ii. ISO 19115: MD_LegalConstraints, MD_SecurityConstraints, CI_ResponsibleParty, CI_Citation, CI_OnlineResouce
iii. SensorML

iv. GML

The best feature is that includes 32 individual (instances) definitions and the weakest part of this ontology is too messy to use it with others applications.

3.1.7 Sensei Observation and Measurement Ontology

The purpose of this ontology is to annotate sensors observation and measurement data. The O&Montology is embedded into the SENSEI[23] resource model. It isn't an active ontology but only a draft version that is based on the OGC's observation and measurement model. Furthermore there are no textual description for this project. As a key framework concepts refers that this project's observations and measurements data can be related to an Entity of Interest (EoI) and this is provided as a Resource also refers that the processes and the services which make the O&M data available provided through a Resource End Point (REP). The SENSEI ontology has a basic hierarchy of concepts and very little property restrictions.As a conclusion the best points of this ontology is that is an observation and a measurement model. On the other hand the weakest points are that the property restrictions are not entirely defined.

3.1.8 Ontonym – Sensor

This ontology has created to represent core conceper in pervasive computing (time, location, people, sensing, provenance, events, device, resource) also the Ontonym is set up of upper ontologies. Sensor is able to describe sensors and the data that they generate. This ontology had created recently but with a lot of upkeep in the future.

Theontologyisorganizedaroundthreekeyconcepts:

i. Sensor: High level description of a sensor and its capabilities (frequency, coverage, accuracy and precision pairs).
ii. Sensor data: Description of sensor observations (observation-specific information, metadata, sensor, timestamp, time period over which the value is valid, rate of change).
iii. These ontologies must be extended to characterize any specific sensor and its data.

In general the ontology is good because ground on existing ontologies and theories which represent time, location, people and events. But it is not recommended for the SSN ontology.

3.1.9 SEEK OBOE

The SEEK Extensible Observation Ontology (OBOE)[24]was developed for the SEEK project, and has since been used by Spire and now is kept by the Scientific Observation Networks-SONet.

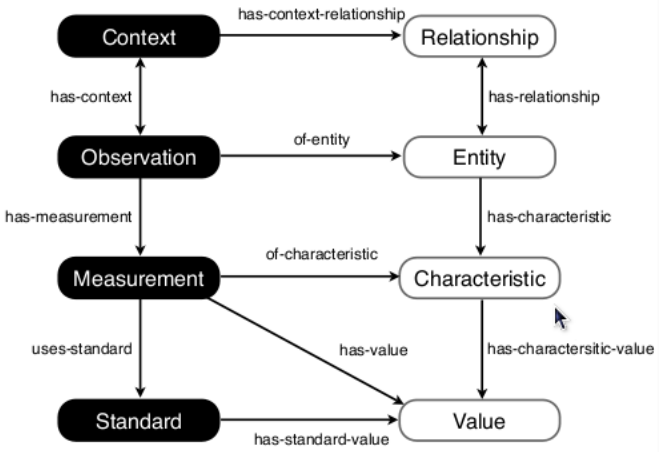


Figure 7. Overview of the OBOE ontology [25]

- The ontology separates observations from the entity being observed: the observation has a measurement while the entity has characteristics, and the measurement is then of that characteristic.
- Entities are extension points into domain models.
- Observations can occur within a context, which in turn is an observation; this property is transitive.

It is not a device ontology but an observation with more than a basic hierarchy of concepts. This ontology is the right match for the SSN ontology given other design requirements (OGC alignment, dovetailing with the device ontology, etc.). In conclusion further discussion is required to find if it is a better approach than that in e.g. the OGC O&M model.

3.1.10 Stimuli-Centered

The Stimouli-Centered [27]ontology created to make a bridge the *sensor-centric* Sensor Model Language (SensorML) and the *user-centric* Observations & Measurements (O&M) specification by focusing on stimuli as objects of sensing. There is no need to have a strong link between sensors and *features of interests* such as O&M also humans can also be used as sensors which are the key to volunteered geographic information. It is an active ontology but it is not complete in the sense of listing sensor(types) or observations but focuses on establishing a common ground for both. The key frameworks are:

i. Stimuli
ii. Observations
iii. Sensors

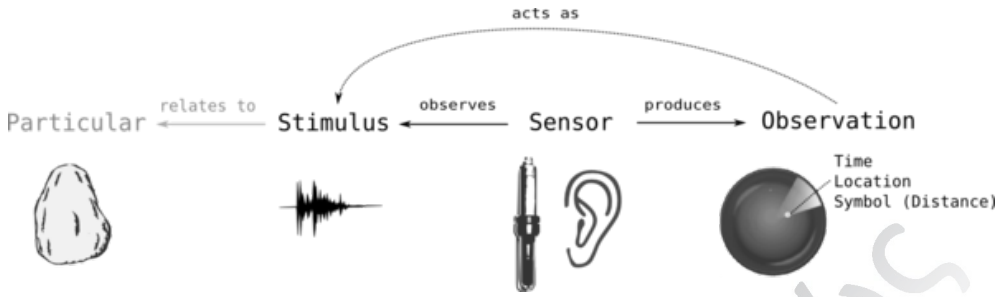


Figure 8. Stimuli as a proxy between a sensor and the object of sensing[28]

This ontology is focused on providing a top-level view on observation and not on specific sensors also provides algebraic specifications using the Haskell[29] programming language to define the core concepts. It depends on SensorML, O&M. The good is that the ontology be used as top-level for the SSN ontology but it needs an OWL version to work with the W3C SSN Ontology.

3.1.11 Socio-Ecological Research and Observation ontology (SERONTO)

This project created for the context of Long Term Ecological Research (LTER) [30]and semantic data integration (ALTER-NET)[31]. The ALTER-NETproject since in March 2009. Currently is responding the 2nd version of development, it's almost finished except two unsatisfiable classes. There're multiple documents which details the ontology creation process which has been used. Thekeyframeworkare:

i. Has Device a investigation_device (one class not further defined by the ontology).
ii. has a *parameter_method* (combo of method and parameter description, including units).
iii. hasInvestigationItem*physical_thing* which is a real world object (SamplingFeature).

Although this ontology is pretty sophisticated. It looks pretty good in the UML and also look fine in Protégé (but with not much classes). This ontology has adopted by a large list of contributors. There are many examples of domain ontologies where are completing the core.

Some of the best features are:

i. The Bio-Diversity use case that is covered by the O&M ontology so anything is handled by SORENTO and OBOE is taken in consideration.
ii. Many generic aspects of an observation are covered in details.
iii. SERONTO imports a unit ontology which looks okay.

And some of the weakest features are:

i the use of subPropertyOf to group properties by themes.

ii The presence of unsatisfiable concepts
iii the choice of using owl:individual (see Individuals by Classes tab in Protege)
    - This is both good
        - Because it grounds the abstract classes into complementary definitions
    - And bad
        - these definitions are less self-descriptive: the end user needs to read the comments (and they are not well documented elsewhere)
        - this design choice could also be a roadblock for possible extensions.

3.2 Distributed Coordination

Nowadays the new technology and the current lifestyle commands more and more communicating systems. For that reason the communications systems are constantly growing continuously and as a result is growing the complicity too. This means that we have exponential growth in the number of communicating elements and the number of communicating subgroups. A characteristic example of this challenge is the evolution of clustering and group communication in enterprise systems. The group communication is about who is in the group and how reliable is the communication between the members. Through the development of systems we passed by master-slave replication to synchronous replication in groups and finally protocols that allow asynchronous progress, such as Virtual Synchrony.

In a virtually synchronous environment processes are allowed to be structured into process groups. Also can make events for example it can face a group like an entity and broadcast to them events, group membership changes, and even migration of an activity from one place to another appear to occur instantaneously -- in other words, synchronously. The scalability of those protocols was restricted by strong consistency semantics.

As the systems became even bigger in size, there were introduced peer-to-peer techniques and gossip protocols were imported, so to relax the consistency semantic, increasing the scale by an order of magnitude. A peer-to-peer (P2P) network refers that is a type of decentralized and distributed network architecture. In this network there’re exists individual nodes ,those nodes called "peers", and they act both as suppliers and as consumers of resources, in contrast to the centralized client–server model where client nodes request access to resources provided by central servers. As gossip protocol refers that is a style of computer-to-computer communication protocol inspired by the form of gossip seen in social networks. Gossip protocols used by modern distributed systems to solve problems that might be difficult to solve in other ways, either because the underlying network has an

inconvenient structure, is extremely large, or because gossip solutions are the most efficient ones available. The peer-to-peer protocol will be described further on chapter 2.2.1.

Continuously Chubby[32] and ZooKeeper[33] is another aspect for managing large distributed systems. In this aspect all the group participants connect to a distributed hub and coordinate their actions through it. ZooKeeper-style services were developed to support the needs of mobile users and mobile applications as a consequence of soaring growth of these.

3.2.1 Peer-To-Peer Protocols

Peer-to-peer (P2P) technology has attracted significant interest in recent years. It's used to share music e.g. Napster, for data storage e.g. Freenet and for scientific computing projects such as SETI@home. First has been used mostly to exchange music files between user's computers (Napster).

Almost a decade later, P2P, technology has been used furthermore than the simple music sharing, anonymous data storage or scientific computing and now begin to gather significant research attention and increasingly widespread use in software communities and industries alike. Scientist, companies and open-software organizations use BitTorrent to distribute bulk data such as software updates, data sets, and media files to many nodes. For example 5 commercial P2P software allows to distribute news and events to their employees, 22 million of people use Skype to communicate every day, there are hundreds of TV channels using live streaming.

But the term P2P has been defined in different ways. Let's clarify what P2P really mean. The P2P is a distributed system with the following properties:

High degree of decentralization.

The peers implement both client and server functionality and most of the system's state and tasks are dynamically allocated among the peers. There are few if any dedicated nodes with centralized state. As a result, the bulk of the computation, bandwidth, and storage needed to operate the system are contributed by participating nodes.

Self-organization.

Once a node is introduced into the system (typically by providing it with the IP address of a participating node and any necessary key material), little or no manual configuration is needed to maintain the system.

Multiple administrative domains.

The participating nodes are not owned and controlled by a single organization. In general, each node is owned and operated by an independent individual who voluntarily joins the system.

Furthermore there are several distinctive specifications about P2P that make them interesting such as:

Low barrier to deployment.

Because P2P systems require little or no dedicated infrastructure, the upfront investment needed to deploy a P2P service tends to be low when compared to client-server systems.

Organic growth.

Because the resources are contributed by participating nodes, a P2P system can grow almost arbitrarily without requiring a"fork-lift upgrade" of existing infrastructure, for example, the replacement of a server with more powerful hardware.

Resilience to faults and attacks.

P2P systems tend to be resilient to faults because there are few if any nodes that are critical to the system's operation. To attack or shut down a P2P system, an attacker must target a large proportion of the nodes simultaneously.

Abundance and diversity of resources.

Popular P2P systems have an abundance of resources that few organizations would be able to afford individually. The resources tend to be diverse in terms of their hardware and software architecture, network attachment, power supply, geographic location and jurisdiction. This diversity reduces their vulnerability to correlated failure, attack, and even censorship.

We'll analyze the most important techniques that make P2P systems works. We discuss fundamental architectural choices like the degree of centralization and the structure of the overlay network. One of the challenges is to built an overlay with a routing capability that works well in the presence of a high membership turnover, this is typically deployed P2P systems. We then present solutions to specific problems addressed in the context of P2P systems: application state maintenance, application-level node coordination, and content distribution.

Πανεπιστήμιο Πειραιώς

3.2.2 Degree of centralization

We can categorize the architecture of P2P systems according to the presence or the absence of centralized components in the system. Partly centralized P2P systems have dedicated controller node that maintains the se of participating nodes and controls system. For example, Napster had a Web site that maintained the membership and content index and Skype has a central site provides log-in, account management and payment.

Resource-intensive operations like transmitting content or computing applications does not involve the controller. Party P2P systems though can provide organic growth and abundant resources.

However, these type of P2P does not offer the same scalability and resilience because of the controller forms a potential bottleneck and a single point of failure and attack. Party centralized P2P systems are relatively simple though and can be managed by a single organization via a controller.

The decentralized P2P system have no dedicated nodes that are can be critical for the operation of the system. Also they haven't inherent bottlenecks and can potentially scale very well. Furthermore, the lack of dedicated nodes makes the resilient to failure, attack and legal challenge. Though in some of those decentralized systems, nodes with plenty of resources, high availability and a publicity routable IP address act as super-nodes. Those nodes have more responsibilities, such as acting a rendezvous point for nodes behind firewalls. Also can increase the efficiency of a P2P system, but this can be its vulnerability to node failure.

3.2.3 Distributed Coordination

Often a group of nodes must coordinate their actions without the central control. For example, the set of nodes that replicated a particular object must inform each other of updates to the object, or a node that is interested in receiving a particular streaming content channel may wish to find, among the nodes that currently receive that channel, one that is nearby and has available upstream network bandwidth. We'll look these two distinct approaches to this problem: epidemic techniques where information spreads virally through the system, and tree-based techniques where distribution trees are formed to spread the information.

We'll focus only on decentralized overlays, coordination can be accomplished by the controller node in partly centralized systems.

- **Unstructured overlays.**

Here the coordination typically relies on epidemic techniques. In these protocols, information is spread through the overlay in a manner similar to the way an infection

spreads in a population: the node that produced the information sends it to (some of) its overlay neighbors, who send it to (some of) their neighbors, and so on. This method of dissemination is very simple and robust. As in all epidemic techniques, there is a trade-off between the speed of information dissemination and overhead. Moreover, if a given piece of information is of interest only to a subset of nodes and these nodes are widely dispersed within the overlay, then the information ends up being needlessly delivered to all nodes. A more efficient way to coordinate the actions among a group of nodes is to form a spanning tree among the nodes. The spanning tree is embedded in the overlay graph, using a decentralized algorithm for spanning tree formation. This tree can then be used to multicast messages to all members, or to compute summaries (for example, sums, averages, minima, or maxima) of state variables within the group. However, this added coordination efficiency must be balanced against the overhead of maintaining the spanning tree in the unstructured overlay network.

- **Structured overlays.**

In structured overlays, spanning trees among any group of overlay nodes can be formed and maintained very efficiently using the KBR [33] primitive making trees the preferred method of coordination in these overlays. To join a spanning tree, a node uses KBR to route to a unique key associated with the group. The resulting union of the paths from all group members form a spanning tree rooted at the node responsible for the group's key. This KBR tree is then used to aggregate and disseminate state associated with the group, and to implement multicast and any cast.

Finally the epidemic techniques typically used for coordination in unstructured overlays are simple and robust to overlay churn, but they may not scale to large overlays or large numbers of groups, and information tends to propagate slowly. Spanning trees can increase the efficiency of coordination, but maintaining a spanning tree in an unstructured overlay adds costs. The additional overhead for maintaining a structured overlay is proportional to the churn in the total overlay membership. Once that overhead is paid, KBR trees enable efficient and fast coordination among potentially numerous, large and dynamic subgroups within the overlay.

Πανεπιστήμιο Πειραιώς

3.2.4 Chubby and ZooKeeper

Chubby is a lock service which is designed for use within a loosely-coupled distributed system consisting of large numbers of small machines connected by a high-speed network.

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services. All of these kinds of services are used in some form or another by distributed applications. Zookeeper is a close clone of Chubby designed to fulfill many of the same roles for HDFS [34] and other Hadoop infrastructure. Before Zookeeper developed in a distributed environment we have taken into account how available and scalable is the service. The Chubby for Google to guarantee a high level of availability runs on minimum five machines and simultaneously provides a built-in master election and failover mechanisms. Considering a cluster of Zookeeper servers (only one of them act as a leader) whose role is to accept and coordinate all writes. The rest of servers are read-only replicas of the master and are direct. This means that if the master break down, immediately any of the other server way, if the master goes down, any other server can take over and serve requests. We have to say that is remarkable that Zookeeper allows the standby servers to serve reads opposed to Chubby. The only disadvantage of this that every cluster node is an exact replica – there is no sharing and the capacity of the service is limited by the size of an individual machine. Two characteristic examples of this are Thialfi from Google and Mobius from Yahoo.

3.2.5 Thialfi: A Client Notification Service for Internet-Scale Applications

Google developed Thialfi [35] for user-facing applications with hundreds of millions of users and billions of objects thus providing sub-second notification delivery in common case and clear semantics despite failures even of entire data centers. All these supports applications written in many languages such as Java, JavaScript and C++ which running in different platforms such as desktops, web browser and mobile phones. To have reliability with Thialfi we have to rely on clients to drive recovery operations avoiding the need for hard state at the server. All these models shared data as versioned objects which are stored at a data center and cached at clients. All clients register with Thialfi to be informed when an object changes and also the application servers notify when updates occur. To synchronize Thialfi data with application servers we have to register clients so Thialfi spread notifications to register them. An attribute of Thialfi is the reliability in the presence of a wide variety of faults. The system ensures their clients that eventually learn of the latest version

of each registered object even if the clients were unreachable at the time the update occurred.

3.2.6 Mobius: Unified Messaging and Data Serving for Mobile Apps

Mobius is a consolidated messaging and data serving system for mobile apps that addresses the challenges, is motivated by data consumption, creation, sharing and messaging requirements of current and future mobile apps. Mobius provides all these features through the MUD (Messaging Unified with Data) abstraction. Mobius consists of both client-side and back-end (cloud) infrastructure.

3.2.7 Public/subscribe(pub/sub)

A fundamental communication service that needs to be offered for efficient group communication is multicast with the popular abstraction public/subscribe(pub/sub) messaging. Publish/subscribe is a distributed interaction example well adapted to the growth of scalable and loosely coupled systems. To compare distributed abstractions, they have introduced a classification based on three dimensions: the decoupling in time, space, and synchronization between producers and consumers of information. Decoupling enforces scalability at the abstraction level, by allowing participants to operate independently of one another. The basic system model for publish/subscribe interaction relies on an event notification service providing storage and management for subscriptions and efficient delivery of events. Such an event service represents a neutral mediator between publishers, acting as producers of events, and subscribers, acting as consumers of events. The publishers publish events through an event service and the subscribers get these events indirectly through the event service. The publishers do not usually hold references to the subscribers, neither do they know how many of these subscribers are participating in the interaction. Similarly, subscribers do not usually hold references to the publishers, neither do they know how many of these publishers are participating in the interaction.

Πανεπιστήμιο Πειραιώς

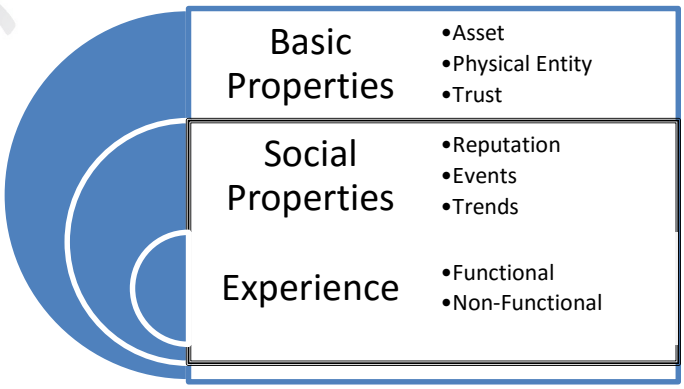
4. Approach

4.1 Ontologies

In previous chapter we have analyzed some ontologies which are integral part of the Internet of Things (IoT).One of those ontologies that mentioned earlier is the Sensor Model Language (SensorML) and will help us to describe and analyze our problem. The primary target of this ontology (SensorML) is to provide a semantically-tied means so we can define the processes and processing components that are associated with the measurement and post-measurement transformation of observations. This includes sensors and actuators as well as computational processes applied pre- and post-measurement. The main concern is to enable the interoperability, initially at the syntactic level and after that at the semantic level (this will happen using ontologies and semantic mediation), in that way the sensors and the processes will be easily understood from machines, and will utilized automatically in complex workflows, and will easily shared between intelligent sensor web nodes. The advantages of the selected ontology (SensorML) is that there are options for positions and dynamic states such as locations, orientations, velocity etc. and there is also an option for real-time access to values and data streams.

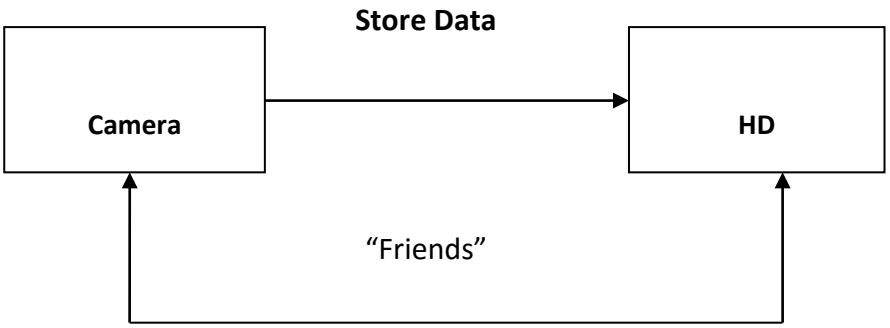
With the aid of SensorML we managed to give some certain characteristics to objects, which are helping them to get the necessary experience. Based on existing attributes we added some additional attributes the social properties, these are helping the objects with their experience. Some of the social properties that mentioned above are:

- Experience
- Goals
- Relationships etc.



The scheme above describes the approach of our work. The basic properties are properties that already existing and we have included for example physical entity, trust, asset etc. In our example we extended those basic properties and we added

the social properties such as reputation, events and trends. With those social properties we have created the idea of “facebook” of things



In the example above we see this idea where two objects became “friends” by sharing experience between them. So the point is to create a society of things which can communicate between them and exchange information.

In the next chapter we will analyze them with the help of SensorML.

4.2 Distributed Coordination

The growth in the number of communicating elements and the number of communicating subgroups is huge. A characteristic example of this challenge is the evolution of clustering and group communication in enterprise systems. Our concept is to bilk from the model of hierarchical structure and create an autonomous system. The idea of the main entity does not exist and the other entities act autonomous when they get the appropriate experience. In other words we will not have an entity which acts like a “leader” and give directions and information to sub entities because every entity will act autonomous when it will have all the information. All of these make our system faster and useful without collisions.

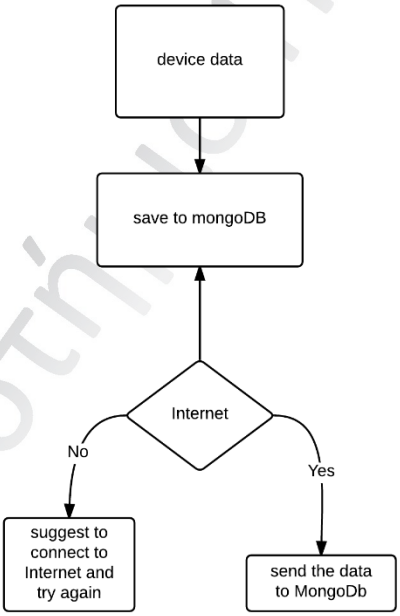
In order to understand how the distributed coordination model works we can refer to the human. When a child is born it isn’t able to speak, to eat by its own, to do things generally and off course this child will necessary need the help of his parents in order to grow and learn things. But as the child grows will learn to eat, to speak and will be able to do more and more things every day by its own, as a result the help of his parents is unnecessary. Like the child at the first the distributed coordination model will need the help of the users and it won’t be able to act entirely autonomous. As the system grows and gather experiences from the user is acting more and more autonomous just like the adult human. At the end the sensor or the application is running without any help from the central system, but will follow the acts from the users.

5. The application

In this chapter we will explain the structure of our applications and also we will provide you some of the code. I order to build those application we've used a really popular program the "eclipse", and as a database we've used the MongoDB. The MongoDB in our case is running in the cloud so we use it as a service from the web site http://www.mongolab.com.

5.1 Flowchart of the first application

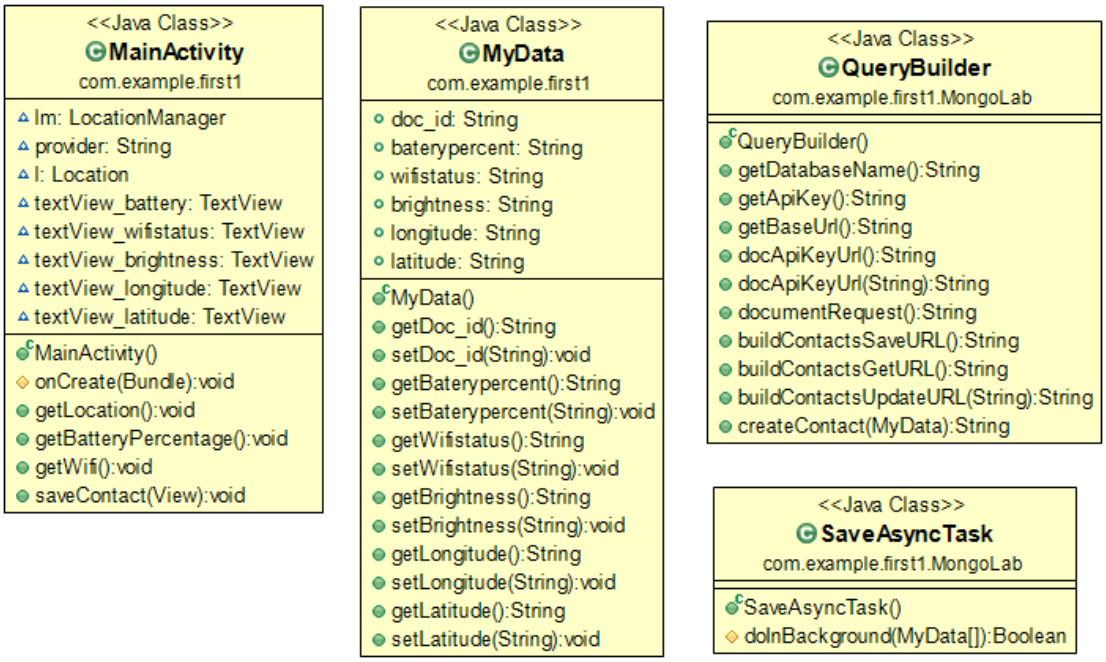
The first application is receiving data from the device, in our case those data are: the battery, the Wi-Fi state (enabled, enabling, disabled, disabling), the current brightness of the screen (in scale from 0 to 255), the longitude and finally latitude. The flowchart from our first application is:



5.2 UML of the first application

In this chapter we'll show the UML representation of our application. The MainActivity class is developed as the main class of this application, the purpose is to collect the data from the device. It has one button to save the data to MongoLAB. The MyData class is creating the variables that we want to pass. The QueryBuilder class is responsible to make the querry to mongoDB right. Finally the saveAsyncTask

is running on the background and is responsible for make the connection to database sending the queries too.



5.3 The implementation of first application

So in this chapter we will explain the implementation of the second application.

5.3.1 MainActivity.java

First we create the MainActivity.java in this activity we get the battery, the Wi-Fi state, the current brightness, longitude and latitude and when we press the button "save to MongoDB" the application check if there is connectivity to internet and depending the result shows the proper message to user.

```
package com.example.first1;

import java.net.UnknownHostException;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.net.ConnectivityManager;
```

```
import android.net.wifi.WifiManager;
import android.os.BatteryManager;
import android.os.Bundle;
import android.provider.Settings.SettingNotFoundException;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import com.example.first1.MongoLab.SaveAsyncTask;

publicclass MainActivity extends Activity {
//We define the variables for the Location Manager, Location and TextView
LocationManager lm;
String provider;
Location l;

TextView textView_battery, textView_wifistatus, textView_brightness,
textView_longitude, textView_latitude;

protectedvoid onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

textView_battery = (TextView) findViewById(R.id.textView_battery);
textView_wifistatus = (TextView) findViewById(R.id.textView_wifistatus);
textView_brightness = (TextView) findViewById(R.id.textView_brightness);
textView_longitude = (TextView) findViewById(R.id.textView_longitude);
textView_latitude = (TextView) findViewById(R.id.textView_latitude);

getBatteryPercentage();
getWifi();
getLocation();

try {
int curBrightnessValue=android.provider.Settings.System.getInt(
getContentResolver(), android.provider.Settings.System.SCREEN_BRIGHTNESS);
textView_brightness.setText(""+ curBrightnessValue);

} catch (SettingNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

publicvoid getLocation(){
lm=(LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
Criteria c=new Criteria();
```

```
provider=lm.getBestProvider(c, false);
l=lm.getLastKnownLocation(provider);
if(l!=null)
{
double lat=l.getLatitude();
double lng=l.getLongitude();

textView_longitude.setText(""+lng);
textView_latitude.setText(""+lat);
}
else
{
textView_longitude.setText("No Provider");
textView_latitude.setText("No Provider");
}
}


publicvoid getBatteryPercentage(){
BroadcastReceiver batteryLevelReceiver = new BroadcastReceiver() {
publicvoid onReceive(Context context, Intent intent) {
context.unregisterReceiver(this);
int currentLevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
int level = -1;
if (currentLevel >= 0 && scale > 0) {
level = (currentLevel * 100) / scale;
}
String tmpStr10 = String.valueOf(level);
textView_battery.setText(tmpStr10);
}
};
IntentFilter batteryLevelFilter = new
IntentFilter(Intent.ACTION_BATTERY_CHANGED);
registerReceiver(batteryLevelReceiver, batteryLevelFilter);
}

publicvoid getWifi(){
BroadcastReceiver WifiStateChangedReceiver = new BroadcastReceiver() {
publicvoid onReceive(Context context, Intent intent) {

int extraWifiState = intent.getIntExtra(WifiManager.EXTRA_WIFI_STATE ,
WifiManager.WIFI_STATE_UNKNOWN);

switch(extraWifiState){
case WifiManager.WIFI_STATE_DISABLED:
```

Πανεπιστήμιο Πειραιώς

```
textView_wifistatus.setText("Disabled");
break;
case WifiManager.WIFI_STATE_DISABLING:
textView_wifistatus.setText("Disabling");
break;
case WifiManager.WIFI_STATE_ENABLED:
textView_wifistatus.setText("Enabled");
break;
case WifiManager.WIFI_STATE_ENABLING:
textView_wifistatus.setText("Enabling");
break;
case WifiManager.WIFI_STATE_UNKNOWN:
textView_wifistatus.setText("Unknown");
break;
}
};
};

IntentFilter wifiLevelFilter = new
IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION);
registerReceiver(WifiStateChangedReceiver, wifiLevelFilter);
}

publicvoid saveContact(View v) throws UnknownHostException {

ConnectivityManager connec =
(ConnectivityManager)getSystemService(getBaseContext().CONNECTIVITY_SERVICE);

// Check for network connections
if ( connec.getNetworkInfo(0).getState() ==
android.net.NetworkInfo.State.CONNECTED ||
connec.getNetworkInfo(0).getState() ==
android.net.NetworkInfo.State.CONNECTING ||
connec.getNetworkInfo(1).getState() ==
android.net.NetworkInfo.State.CONNECTING ||
connec.getNetworkInfo(1).getState() == android.net.NetworkInfo.State.CONNECTED
) {

//Concert the text from TextView to String
MyData data = new MyData();
data.baterypercent = textView_battery.getText().toString();
data.wifistatus = textView_wifistatus.getText().toString();
data.brightness = textView_brightness.getText().toString();
data.longitude = textView_longitude.getText().toString();
data.latitude = textView_latitude.getText().toString();

//Run the SaveAsyncTask so we can send the data to cloud
```

```
SaveAsyncTask tsk = new SaveAsyncTask();
tsk.execute(data);


Toast.makeText(this, " Data send. Thank you for your contribution!",
Toast.LENGTH_LONG).show();
//Check for network connections
} elseif (
connec.getNetworkInfo(0).getState()                                    ==
android.net.NetworkInfo.State.DISCONNECTED ||
connec.getNetworkInfo(1).getState()                                    ==
android.net.NetworkInfo.State.DISCONNECTED  ) {

Toast.makeText(this, " I'm sorry you I need Internet to send my DATA. Please
connect me to Internet and try again! ", Toast.LENGTH_LONG).show();

}}}
```

5.3.2 MyData.java

In the file MyData.java we declared the variables that we want to send to cloud in our case at Mongodb.

```
package com.example.first1;

publicclass MyData {

public String doc_id;
public String baterypercent;
public String wifistatus;
public String brightness;
public String longitude;
public String latitude;

public String getDoc_id() {
returndoc_id;
}
publicvoid setDoc_id(String doc_id) {
this.doc_id = doc_id;
}
public String getBaterypercent() {
returnbaterypercent;
}
publicvoid setBaterypercent(String baterypercent) {
this.baterypercent = baterypercent;
}
```

Πανεπιστήμιο Πειραιώς

```
public String getWifistatus() {
returnwifistatus;
}
publicvoid setWifistatus(String wifistatus) {
this.wifistatus = wifistatus;
}
public String getBrightness() {
returnbrightness;
}
publicvoid setBrightness(String brightness) {
this.brightness = brightness;
}
public String getLongitude() {
returnlongitude;
}
publicvoid setLongitude(String longitude) {
this.longitude = longitude;
}
public String getLatitude() {
returnlatitude;
}
publicvoid setLatitude(String latitude) {
this.latitude = latitude;
}}
```

5.3.3 QueryBuilder.java

In this file we define the database that we've created in the web-site www.mogolab.com, the API key, the database that will allow us to manage our database, the docid and finally we created the query.

```
package com.example.first1.MongoLab;

import com.example.first1.MyData;

publicclass QueryBuilder {

/**
* Specify the database name
*/
public String getDatabaseName() {
return"code102";
}
```

Πανεπιστήμιο Πειραιώς

```
/**
* Specify the MongoLab API key
*/
public String getApiKey() {
return"K6onPQmCPU5S1N9PqztFmmzMlMKvJpRp";
}

/**
* This constructs the URL that allows to manage your database,
* collections and documents
*/
public String getBaseUrl()
{
returnhttps://api.mongolab.com/api/1/databases/"+getDatabaseName()+"/collectio
ns/";
}

/**
* Completes the formating of your URL and adds the API key at the end
*/
public String docApiKeyUrl()
{
return"?apiKey="+getApiKey();
}

/**
* Get a specified document with the docid
*/
public String docApiKeyUrl(String docid)
{
return"/"+docid+"?apiKey="+getApiKey();
}

/**
* Returns the docs101 collection
*/
public String documentRequest()
{
return"docs101";
}

/**
* Builds a complete URL using the methods specified above
*/
public String buildContactsSaveURL()
{
return getBaseUrl()+documentRequest()+docApiKeyUrl();
```

Πανεπιστήμιο Πειραιώς

```
}

/**
* This method is identical to the one above.
*/
public String buildContactsGetURL()
{
return getBaseUrl()+documentRequest()+docApiKeyUrl();
}

/**
* Get a Mongodb document that corresponds to the given object id parameter doc_id
*/
public String buildContactsUpdateURL(String doc_id)
{
return getBaseUrl()+documentRequest()+docApiKeyUrl(doc_id);
}

/**
* Formats the contact details for MongoHLab Posting data: battery, wifistatus, brightness, longitude, latitude
*/
public String createData(MyData data)
{
return String
.format("{\"battery_percent\": \"%s\",\"wifi_status\": \"%s\",\"brightness\": \"%s\",\"longitude\": \"%s\",\"latitude\": \"%s\"}",
data.baterypercent, data.wifistatus, data.brightness, data.longitude, data.latitude);
}
}
```

5.3.4 SaveAsyncTask.java

In this file we created the AsyncTask that connects to MongoDB and execute the query. The AsycnTask runs in background.

```
package com.example.first1.MongoLab;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;

import android.os.AsyncTask;
```

Πανεπιστήμιο Πειραιώς

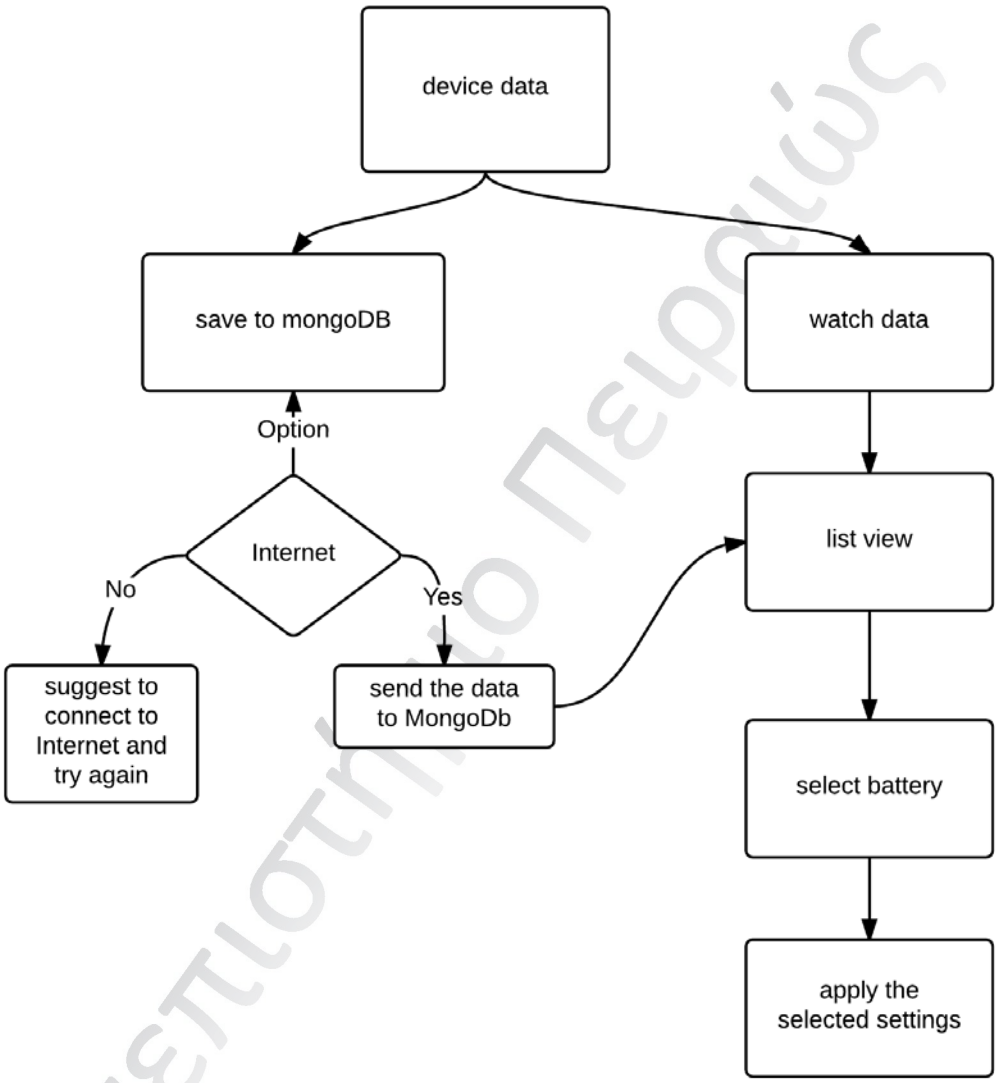
```
import com.example.first1.MyData;;

public class SaveAsyncTask extends AsyncTask<MyData, Void, Boolean> {

@Override
protected Boolean doInBackground(MyData... arg0) {
try
{
MyData contact = arg0[0];

QueryBuilder qb = new QueryBuilder();

HttpClient httpClient = new DefaultHttpClient();
HttpPost request = new HttpPost(qb.buildContactsSaveURL());

StringEntity params =new StringEntity(qb.createContact(contact));
request.addHeader("content-type", "application/json");
request.setEntity(params);
HttpResponse response = httpClient.execute(request);

if(response.getStatusLine().getStatusCode()<205)
{
return true;
}
else
{
return false;
}
} catch (Exception e) {
//e.getCause();
String val = e.getMessage();
String val2 = val;
return false;
}}}
```

5.4 Information and service flow of the second application

In this application we also check the device data and we have two options. The first options is to press the button “save to mongoDB”, the application check the connectivity to network and if the device is connected to internet sends the data to cloud. If the device is not connected to internet the application shows to user and urge the user to try again. The second option is to see the data that already exists on database so we press the button “watch data”. The application show us the data in a list view (the list view shows the battery of each query of the database). The user can

Πανεπιστήμιο Πειραιώς

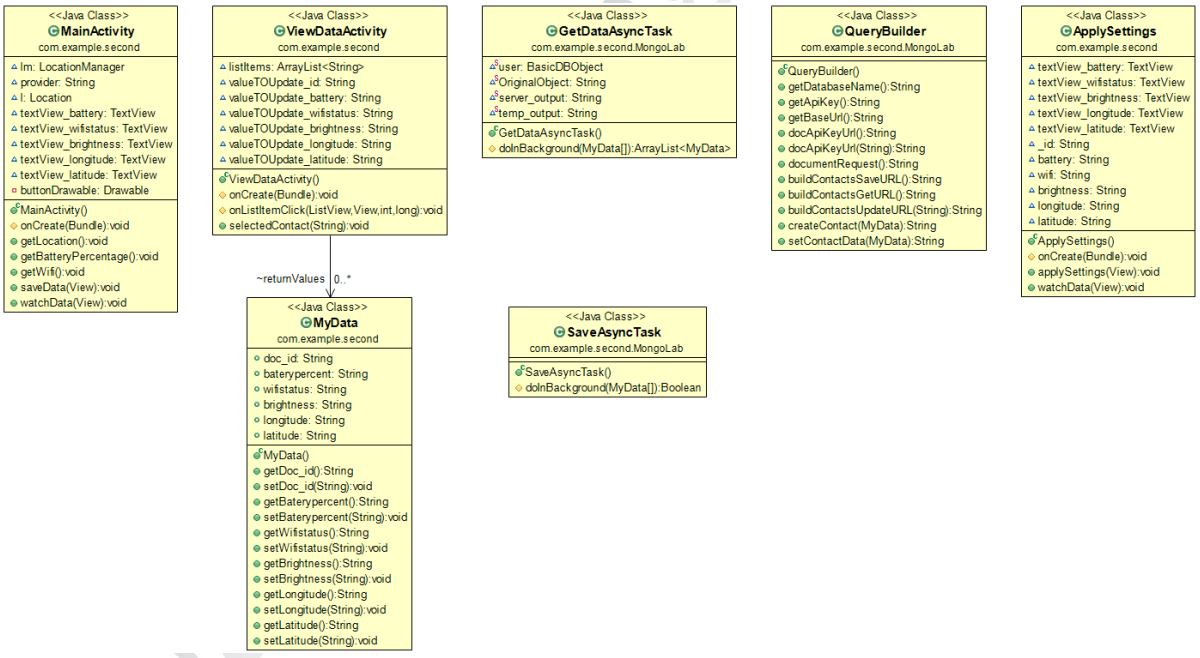
select a value from the above list. The selected values will show on the next screen and the user can apply the brightness of the screen or can watch the data again from the beginning.



Πανεπιστήμιο Πειραιώς

5.5 UML from the second application

The MainActivity class is the first class that opens. This class collect all the data from the device and give the user the option to save the data or to watch the saved data. The ViewDataActivity class is showing on a list view the results of the database, on our case mongoLAB. The GetDataAsyncTask class help us get the data from the cloud in our case the mongoLAB. The SaveAsyncTask is saving the collected data to cloud. The QueryBuilder is the constrictor of the basic query that we are sending to cloud, also in this file we configure the database we want to save the data. The ApplySettings class is showing to us the data we've selected from the list (ViewDataActivity) and give the user two options. The first option is to apply the selected data and the second is to go again and watch the data so user goes back to ViewDataActivity.



5.6 The implementation of the second application

So in this chapter we will explain the implementation of the second application.

5.6.1 MainActivity.java

This class is identical with the first application, but we have added the button that link us to ViewDataList class.

```
package com.example.second;

import java.net.UnknownHostException;
import android.app.Activity;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.graphics.drawable.Drawable;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.net.ConnectivityManager;
import android.net.wifi.WifiManager;
import android.os.BatteryManager;
import android.os.Bundle;
import android.provider.Settings.SettingNotFoundException;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.example.second.MongoLab.SaveAsyncTask;

public class MainActivity extends Activity {
//Define variables
LocationManager lm;
String provider;
Location l;
TextView textView_battery, textView_wifistatus, textView_brightness,
textView_longitude, textView_latitude;

private Drawable buttonDrawable;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

//Find the TextView of activity_main.xml
textView_battery = (TextView) findViewById(R.id.textView_battery);
textView_wifistatus = (TextView) findViewById(R.id.textView_wifistatus);
textView_brightness = (TextView) findViewById(R.id.textView_brightness);
textView_longitude = (TextView) findViewById(R.id.textView_longitude);
textView_latitude = (TextView) findViewById(R.id.textView_latitude);
```

Πανεπιστήμιο Πειραιώς

```
getBatteryPercentage();
getWifi();
getLocation();

try {
int curBrightnessValue=android.provider.Settings.System.getInt(
getContentResolver(), android.provider.Settings.System.SCREEN_BRIGHTNESS);
textView_brightness.setText(""+ curBrightnessValue);

} catch (SettingNotFoundException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}

public void getLocation(){
lm=(LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
Criteria c=new Criteria();

provider=lm.getBestProvider(c, false);
l=lm.getLastKnownLocation(provider);
if(l!=null)
{
double lat=l.getLatitude();
double lng=l.getLongitude();

textView_longitude.setText(""+lng);
textView_latitude.setText(""+lat);
}
else
{
textView_longitude.setText("No Provider");
textView_latitude.setText("No Provider");
}
}

//Get the battery percent
public void getBatteryPercentage(){
BroadcastReceiver batteryLevelReceiver = new BroadcastReceiver() {
public void onReceive(Context context, Intent intent) {
context.unregisterReceiver(this);
int currentLevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
int level = -1;
if (currentLevel >= 0 && scale > 0) {
level = (currentLevel * 100) / scale;
```

Πανεπιστήμιο Πειραιώς

```
}

String tmpStr10 = String.valueOf(level);
textView_battery.setText(tmpStr10);
}
};
IntentFilter batteryLevelFilter = new
IntentFilter(Intent.ACTION_BATTERY_CHANGED);
registerReceiver(batteryLevelReceiver, batteryLevelFilter);
}

//Get the wi-fi state
public void getWifi(){
BroadcastReceiver WifiStateChangedReceiver = new BroadcastReceiver() {
public void onReceive(Context context, Intent intent) {

int extraWifiState = intent.getIntExtra(WifiManager.EXTRA_WIFI_STATE ,
WifiManager.WIFI_STATE_UNKNOWN);

switch(extraWifiState){
case WifiManager.WIFI_STATE_DISABLED:
textView_wifistatus.setText("Disabled");
break;
case WifiManager.WIFI_STATE_DISABLING:
textView_wifistatus.setText("Disabling");
break;
case WifiManager.WIFI_STATE_ENABLED:
textView_wifistatus.setText("Enabled");
break;
case WifiManager.WIFI_STATE_ENABLING:
textView_wifistatus.setText("Enabling");
break;
case WifiManager.WIFI_STATE_UNKNOWN:
textView_wifistatus.setText("Unknown");
break;
}
};
};

IntentFilter wifiLevelFilter = new
IntentFilter(WifiManager.WIFI_STATE_CHANGED_ACTION);
registerReceiver(WifiStateChangedReceiver, wifiLevelFilter);
}

//Act when the button is pressed
public void saveData(View v) throws UnknownHostException {
```

Πανεπιστήμιο Πειραιώς

```
ConnectivityManager connec =
(ConnectivityManager)getSystemService(getBaseContext().CONNECTIVITY_SERVICE);

// Check for network connections
if ( connec.getNetworkInfo(0).getState() ==
android.net.NetworkInfo.State.CONNECTED ||
connec.getNetworkInfo(0).getState() ==
android.net.NetworkInfo.State.CONNECTING ||
connec.getNetworkInfo(1).getState() ==
android.net.NetworkInfo.State.CONNECTING ||
connec.getNetworkInfo(1).getState() == android.net.NetworkInfo.State.CONNECTED
) {

MyData data = new MyData();
data.baterypercent = textView_battery.getText().toString();
data.wifistatus = textView_wifistatus.getText().toString();
data.brightness = textView_brightness.getText().toString();
data.longitude = textView_longitude.getText().toString();
data.latitude = textView_latitude.getText().toString();

SaveAsyncTask tsk = new SaveAsyncTask();
tsk.execute(data);

Toast.makeText(this, " Data send. Thank you for your contribution!",
Toast.LENGTH_LONG).show();
Intent i = new Intent(this, ViewDataActivity.class);
startActivity(i);
}
// Check for network connections
else if (
connec.getNetworkInfo(0).getState() ==
android.net.NetworkInfo.State.DISCONNECTED ||
connec.getNetworkInfo(1).getState() ==
android.net.NetworkInfo.State.DISCONNECTED  ) {

Toast.makeText(this, " I'm sorry you I need Internet to send my DATA. Please
connect me to Internet and try again! ", Toast.LENGTH_LONG).show();
}}

//Act when the button is pressed
public void watchData(View n) throws UnknownHostException{

Intent i = new Intent(this, ViewDataActivity.class);
startActivity(i);
}}
```

Πανεπιστήμιο Πειραιώς

5.6.2 MyData.java

This java class is similar from the first application.

```
package com.example.second;

publicclass MyData {

public String doc_id;
public String baterypercent;
public String wifistatus;
public String brightness;
public String longitude;
public String latitude;

public String getDoc_id() {
returndoc_id;
}
publicvoid setDoc_id(String doc_id) {
this.doc_id = doc_id;
}
public String getBaterypercent() {
returnbaterypercent;
}
publicvoid setBaterypercent(String baterypercent) {
this.baterypercent = baterypercent;
}

public String getWifistatus() {
returnwifistatus;
}
publicvoid setWifistatus(String wifistatus) {
this.wifistatus = wifistatus;
}
public String getBrightness() {
returnbrightness;
}
publicvoid setBrightness(String brightness) {
this.brightness = brightness;
}
public String getLongitude() {
returnlongitude;
}
publicvoid setLongitude(String longitude) {
this.longitude = longitude;
}
```

Πανεπιστήμιο Πειραιώς

```
public String getLatitude() {
returnlatitude;
}
publicvoid setLatitude(String latitude) {
this.latitude = latitude;
}
}
```

5.6.3 ViewDataActivity.java

This java class is the class that create the list view with data provided by the cloud (in our case the service of mongoDB).

```
package com.example.second;

import java.util.ArrayList;
import java.util.concurrent.ExecutionException;
import android.app.ListActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

import com.example.second.MongoLab.GetDataAsyncTask;

/**
* This activity retrieves the mongolab data and displays them in a listview.
*
*/
publicclass ViewDataActivity extends ListActivity{
//Define the Array List
ArrayList<MyData>returnValues = new ArrayList<MyData>();
ArrayList<String>listItems = new ArrayList<String>();

String valueTOUpdate_id;
String valueTOUpdate_battery;
String valueTOUpdate_wifistatus;
String valueTOUpdate_brightness;
String valueTOUpdate_longitude;
String valueTOUpdate_latitude;
```

Πανεπιστήμιο Πειραιώς

```
protectedvoid onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.list_view_data);

//Get cloud data
GetDataAsyncTask task = new GetDataAsyncTask();
try {
returnValues = task.execute().get();
} catch (InterruptedException e) {
e.printStackTrace();
} catch (ExecutionException e) {
e.printStackTrace();
}

for(MyData x: returnValues){
listItems.add(x.getBaterypercent());
}
setListAdapter(new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,
listItems));
}

//Define that to do when a result is pressed
protectedvoid onListItemClick(ListView l, View v, int position, long id) {
super.onListItemClick(l, v, position, id);

String selectedValue = (String) getListAdapter().getItem(position);
Toast.makeText(this, "The selected battery is :"+selectedValue,
Toast.LENGTH_SHORT).show();
selectedContact(selectedValue);

Bundle dataBundle = new Bundle();
dataBundle.putString("_id", valueTOUpdate_id);
dataBundle.putString("battery_percent", valueTOUpdate_battery);
dataBundle.putString("wifi_status", valueTOUpdate_wifistatus);
dataBundle.putString("brightness", valueTOUpdate_brightness);
dataBundle.putString("longitude", valueTOUpdate_longitude);
dataBundle.putString("latitude", valueTOUpdate_latitude);

//Open the ApplySetting class so we can see the results from the data we receives
from cloud
Intent moreDetailsIntent = new Intent(this,ApplySettings.class);
moreDetailsIntent.putExtras(dataBundle);
startActivity(moreDetailsIntent);
}

/*
```

Πανεπιστήμιο Πειραιώς

```
* Retrieves the full details of a selected data.
* The details are then passed onto the Apply Setting class.
*/
publicvoid selectedContact(String selectedValue){
for(MyData x: returnValues){
if(selectedValue.contains(x.getBaterypercent())){
valueTOUpdate_id = x.getDoc_id();
valueTOUpdate_battery = x.getBaterypercent();
valueTOUpdate_wifistatus=x.getWifistatus();
valueTOUpdate_brightness=x.getBrightness();
valueTOUpdate_longitude=x.getLongitude();
valueTOUpdate_latitude=x.getLatitude();
}}}}
```

5.6.4 ApplySettings.java

This class file is similar with the MainActivity.java but we the have insert “if” statements witch decide what to do with brightness and Wi-Fi according to data from the cloud.

```
package com.example.second;

import java.net.UnknownHostException;
import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.os.RemoteException;
import android.view.View;
import android.view.WindowManager;
import android.widget.TextView;
import android.widget.Toast;


public class ApplySettings extends Activity {
TextView textView_battery, textView_wifistatus, textView_brightness,
textView_longitude, textView_latitude;

String _id;
String battery;
String wifi;
String brightness;
String longitude, latitude;
```

Πανεπιστήμιο Πειραιώς

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_apply);
textView_battery = (TextView) findViewById(R.id.textView_battery);
textView_wifistatus = (TextView) findViewById(R.id.textView_wifistatus);
textView_brightness = (TextView) findViewById(R.id.textView_brightness);
textView_longitude = (TextView) findViewById(R.id.textView_longitude);
textView_latitude = (TextView) findViewById(R.id.textView_latitude);

//Obtain details of the clicked data
Bundle getBundle = null;
getBundle = this.getIntent().getExtras();
_id = getBundle.getString("_id");
battery= getBundle.getString("battery_percent");
wifi= getBundle.getString("wifi_status");
brightness=getBundle.getString("brightness");
longitude=getBundle.getString("longitude");
latitude=getBundle.getString("latitude");


textView_battery.setText(battery);
textView_wifistatus.setText(wifi);
textView_brightness.setText(brightness);
textView_longitude.setText(longitude);
textView_latitude.setText(latitude);
}

public void applySettings(View v) throws UnknownHostException {

int bat= Integer.parseInt(battery);
int br = Integer.parseInt(brightness);

Toast.makeText(this, "The battery is: "+bat +"\n The brightness will be: " +br,
Toast.LENGTH_LONG).show();
//adjust brightness and wi-fi according to the queries
if(bat==100)
{
WifiManager wifiManager =
(WifiManager)getBaseContext().getSystemService(Context.WIFI_SERVICE);
wifiManager.setWifiEnabled(true);
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS_MODE, 0);

WindowManager.LayoutParams layoutParams = getWindow().getAttributes();
layoutParams.screenBrightness = br/255.0f;
getWindow().setAttributes(layoutParams);
```

Πανεπιστήμιο Πειραιώς

```
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS, br);
}

else if(bat<100&&bat>75)
{
WifiManager wifiManager =
(WifiManager)getBaseContext().getSystemService(Context.WIFI_SERVICE);
wifiManager.setWifiEnabled(true);
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS_MODE, 0);

WindowManager.LayoutParams layoutParams = getWindow().getAttributes();
layoutParams.screenBrightness = br/255.0f;
getWindow().setAttributes(layoutParams);

android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS, br);
}
else if(bat<75&&bat>50)
{
WifiManager wifiManager =
(WifiManager)getBaseContext().getSystemService(Context.WIFI_SERVICE);
wifiManager.setWifiEnabled(true);
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS_MODE, 0);

WindowManager.LayoutParams layoutParams = getWindow().getAttributes();
layoutParams.screenBrightness = br/255.0f;
getWindow().setAttributes(layoutParams);

android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS, br);
}
else if(bat<50&&bat>25)
{
WifiManager wifiManager =
(WifiManager)getBaseContext().getSystemService(Context.WIFI_SERVICE);
wifiManager.setWifiEnabled(true);
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS_MODE, 0);

WindowManager.LayoutParams layoutParams = getWindow().getAttributes();
layoutParams.screenBrightness = br/255.0f;
getWindow().setAttributes(layoutParams);
```

Πανεπιστήμιο Πειραιώς

```
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS, br);
}
else if(bat<=25)
{
WifiManager wifiManager =
(WifiManager)getBaseContext().getSystemService(Context.WIFI_SERVICE);
wifiManager.setWifiEnabled(false);
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS_MODE, 0);

WindowManager.LayoutParams layoutParams = getWindow().getAttributes();
layoutParams.screenBrightness = br/100.0f;
getWindow().setAttributes(layoutParams);

android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS, br);
}

}

public void watchData(View n) throws UnknownHostException{

Intent i = new Intent(this, ViewDataActivity.class);
startActivity(i);

}
}
```

5.6.5 GetDataAsyncTask

In this file we retrieve the data from mongoLab.

```
package com.example.second.MongoLab;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;

import android.os.AsyncTask;

import com.mongodb.BasicDBList;
```

Πανεπιστήμιο Πειραιώς

```
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;
import com.example.second.MyData;
/**
* Async Task to retrieve data frommongolab
*
*/
publicclass GetDataAsyncTask extends AsyncTask<MyData, Void,
ArrayList<MyData>> {
static BasicDBObject user = null;
static String OriginalObject = "";
static String server_output = null;
static String temp_output = null;

protected ArrayList<MyData> doInBackground(MyData... arg0) {

ArrayList<MyData> mydata = new ArrayList<MyData>();
try
{

QueryBuilder qb = new QueryBuilder();
URL url = new URL(qb.buildContactsGetURL());
HttpURLConnection conn = (HttpURLConnection) url
.openConnection();
conn.setRequestMethod("GET");
conn.setRequestProperty("Accept", "application/json");

if (conn.getResponseCode() != 200) {
thrownew RuntimeException("Failed : HTTP error code : "
+ conn.getResponseCode());
}

BufferedReader br = new BufferedReader(new InputStreamReader(
(conn.getInputStream())));

while ((temp_output = br.readLine()) != null) {
server_output = temp_output;
}

// create a basic db list
String mongoarray = "{ artificial_basicdb_list: "+server_output+"}";
Object o = com.mongodb.util.JSON.parse(mongoarray);


DBObject dbObj = (DBObject) o;
BasicDBList contacts = (BasicDBList) dbObj.get("artificial_basicdb_list");
```

```
for (Object obj : contacts) {
DBObject userObj = (DBObject) obj;
MyData temp = new MyData();
temp.setDoc_id(userObj.get("_id").toString());
temp.setBaterypercent(userObj.get("battery_percent").toString());
temp.setWifistatus(userObj.get("wifi_status").toString());
temp.setBrightness(userObj.get("brightness").toString());
temp.setLongitude(userObj.get("longitude").toString());
temp.setLatitude(userObj.get("latitude").toString());
mydata.add(temp);
}
}catch (Exception e) {
e.getMessage();
}
return mydata;
}
}
```

5.6.6 QueryBuilder.java

This class is similar to the first application and is the responsible to pass important information about the API key of the mongoLab, the collection name, the database name, the way data will save to the mongoLab.

```
package com.example.second.MongoLab;


import com.example.second.MyData;
publicclass QueryBuilder {

/**
* Specify database name here
*/
public String getDatabaseName() {
return"code102";
}

/**
* Specify MongoLab API here
*/
public String getApiKey() {
return"K6onPQmCPU5S1N9PqztFmmzMlMKvJpRp";
}
```

Πανεπιστήμιο Πειραιώς

```
/**
* This constructs the URL that allows you to manage our database, collections and
documents
*/
public String getBaseUrl()
{
return"https://api.mongolab.com/api/1/databases/"+getDatabaseName()+"/collecti
ons/";
}

/**
* Completes the formating of your URL and adds your API key at the end
*/
public String docApiKeyUrl()
{
return"?apiKey="+getApiKey();
}

/**
* Get a specified document, with docid as parameter
*/
public String docApiKeyUrl(String docid)
{
return"/"+docid+"?apiKey="+getApiKey();
}

/**
* Returns the docs101 collection
*/
public String documentRequest()
{
return"docs101";
}

/**
* Builds a complete URL using the methods specified above
*/
public String buildContactsSaveURL()
{
return getBaseUrl()+documentRequest()+docApiKeyUrl();
}

/**
* This method is identical to the one above.
*/
public String buildContactsGetURL()
{
```

Πανεπιστήμιο Πειραιώς

```
return getBaseUrl()+documentRequest()+docApiKeyUrl();
}

/**
* Get a Mongodb document that corresponds to the given object id
*/
public String buildContactsUpdateURL(String doc_id)
{
return getBaseUrl()+documentRequest()+docApiKeyUrl(doc_id);
}


/**
* Formats the data details for MongoHLab Posting
* @return
*/
public String createContact(MyData data)
{
return String
.format("{\"battery_percent\": \"%s\",\"wifi_status\": \"%s\",\"brightness\":
\"%s\",\"longitude\": \"%s\",\"latitude\": \"%s\"}",
data.baterypercent, data.wifistatus, data.brightness, data.longitude, data.latitude);
}

/**
* Update a given data record
* @return
*/
public String setContactData(MyData data) {
return String.format("{ \"$set\" : "
+ "{\"battery_percent\" : \"%s\",\"wifi_status\": \"%s\",\"brightness\":
\"%s\",\"longitude\": \"%s\",\"latitude\": \"%s\"}",
data.getBaterypercent(), data.getWifistatus(), data.getBrightness(),
data.getLongitude(), data.getLatitude());
}}
```

5.6.7 SaveAsyncTask.java

```
package com.example.second.MongoLab;

import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
```

Πανεπιστήμιο Πειραιώς

```
import android.os.AsyncTask;
import com.example.second.MyData;;

public class SaveAsyncTask extends AsyncTask<MyData, Void, Boolean> {

protected Boolean doInBackground(MyData... arg0) {
try
{
MyData contact = arg0[0];

QueryBuilder qb = new QueryBuilder();

HttpClient httpClient = new DefaultHttpClient();
HttpPost request = new HttpPost(qb.buildContactsSaveURL());

StringEntity params =new StringEntity(qb.createContact(contact));
request.addHeader("content-type", "application/json");
request.setEntity(params);
HttpResponse response = httpClient.execute(request);

if(response.getStatusLine().getStatusCode()<205)
{
return true;
}
else
{
return false;
}
} catch (Exception e) {
//e.getCause();
String val = e.getMessage();
String val2 = val;
return false;
}}}
```

5.7 Case study

In order to understand the way the application works we will explain some of the cases. For example the first device sends to mongoLab the following data:

```
battery_percent: 100,
wifi_status: Enabled,
brightness: 255,
longitude: 23.7438103,
latitude: 38.0465187
```

Πανεπιστήμιο Πειραιώς

According to our code if the user of the second application select this query from the list view and push the button “apply settings” the Wi-Fi state will be on and the screen brightness will be 255. Those settings will be applied according to this code.

```
if(bat==100)
{
WifiManager wifiManager =
(WifiManager)getBaseContext().getSystemService(Context.WIFI_SERVICE);
//with this command stays enabled
wifiManager.setWifiEnabled(true);
//with this command we deactivated the auto-brightnes that some devides might have
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS_MODE, 0);
//here we change the brightness to our current window
WindowManager.LayoutParams layoutParams = getWindow().getAttributes();
layoutParams.screenBrightness = br/100.0f;
getWindow().setAttributes(layoutParams);
//finally we certain value of brightness goes to android system
android.provider.Settings.System.putInt(getContentResolver(),
android.provider.Settings.System.SCREEN_BRIGHTNESS, br);
}
```

The following table will demonstrate the case scenarios that will happen in our example.

Baterry (%)	Wifi	Brightness (0-255)
100	enable	255
75-100	enable	200
50-75	enable	150
25-50	disable	100
<=25	disable	50

Πανεπιστήμιο Πειραιώς

5.8 Future work

In the near future, the Internet of Things (IoT) will have been fully developed, everywhere and “things” will be all over the city, all over us. We’ve successfully developed a representative example of sharing experiences, but in the future we could add a lot of features. We could create groups of the “things” so it will be easy sharing experience between two similar “things”, e.g. we could create a group only with public transportation buses, in this group all those buses could exchange data about geolocation, the current capacity of people, the remaining fuel or even if a mechanical problem happened. This could be easily solve the problem of the groups by adopting “friend” feature. Every “thing” could make a friend another “thing” if same features are existing so they can easily exchange experiences by their own. The “friend” feature is handier for the users, so if you have a smart device and become a friend with a similar one you would be able to exchange experiences, and apply those settings.

Πανεπιστήμιο Πειραιώς

6. Conclusion

In a society that development is rapid and people use more and more the Internet and depend on that we came to answer the question how useful is to have a system where all the things can communicate between them and share their experiences. The importance of Internet of Things (IoT) played a very important role in our research because this theory was the base to start the idea of share experience between different things. After a short description about the Internet of Things and what the usage of them in our daily life we described the existing problems that we must solved. The solution from all the above was the ontologies and the distributed coordination. The main ontology that help us to solve and answer the questions was the SensorML. Also with the distributed coordination we achieve the goal to make our application autonomous. With the help from all the above we develop an application in Android that is a representative example of sharing experiences through different things. We used the MongoDB as the main database of our application which in our case is Software as a Service database (SaaS).

Πανεπιστήμιο Πειραιώς

References

[1] Joachim W. Walewski (Siemens), “Initial architectural reference model for IoT”, http://www.iot-a.eu/public/public-documents/d1.2, 2013

[2] J. Gubbi, R. Buyya,S. Marusic,M. Palaniswamia, “Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions”, http://arxiv.org/ftp/arxiv/papers/1207/1207.0203.pdf

[3] Joachim W. Walewski (Siemens), “Initial architectural reference model for IoT”, http://www.iot-a.eu/public/public-documents/d1.2, 2013

[4] Joachim W. Walewski (Siemens), “Initial architectural reference model for IoT”, http://www.iot-a.eu/public/public-documents/d1.2, 2013

[5] Kaushik Das, IPv6, Information about IPv6, http://ipv6.com/articles/general/ipv6-the-next-generation-internet.htm, 2013

[6] Sabita Maharjan, “RFID and IOT: An overview”, Simula Research Laboratory University of Oslo, 2010

[7] C. Alcaraz, P. Najera, J.Lopez, R. Roman, “Wireless Sensor Networks and the Internet of Things: Do We Need a Complete Integration?”, University of Malaga Malaga, Spain

[8] Image from paper with title “Technologies and Architectures of the Internet-of-Things (IoT) for Health and Well-being”

[9] Image for the future smart city, http://www.libelium.com/top_50_iot_sensor_applications_ranking/#show_infographic

[10] About OGC, http://www.opengeospatial.org/standards

[11] Sensor Model Language (SensorML), http://www.opengeospatial.org/standards/sensorml

[12] Cory Henson, http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report#O.26M-OWL_.28SemSOS.29

[13] Resource Description Framework (RDF) , http://www.w3.org/RDF/, 2014

[14] Cory A. Henson, Josh K. Pschorr, Amit P. Sheth, K. Thirunarayan, “SemSOS: Semantic Sensor Observation Service” , Kno.e.sis Center, Department of Computer Science and Engineering Wright State University, Dayton, 2009

[15] D. Connolly, Frank van Harmelen, I. Horrocks, D. L. McGuinness , P. F. Patel-Schneider, L. A. Stein, “DAML+OIL Reference Description”, 2001

[16] Cory A. Henson, Josh K. Pschorr, Amit P. Sheth, K. Thirunarayan, “SemSOS: Semantic Sensor Observation Service” , Kno.e.sis Center, Department of Computer Science and Engineering Wright State University, Dayton, 2009

[17] Raúl García Castro, definition of SDO, http://www.w3.org/2005/Incubator/ssn/wiki/Incubator_Report#SDO

[18] Ian Niles, “Mapping WordNet to the SUMO Ontology”, Teknowledge Corporation

[19] M. Comptona, P. Barnaghib, L. Bermudezc, R. Garc´ıa-Castrod, O. Corchod, S. Coxe, J. Graybeal,M. Hauswirthf, C. Hensong, A. Herzogh, V. Huangi, K. Janowiczj, W. David Kelsey, Danh Le Phuocf,L. Leforta, M. Leggierif, H. Neuhaus, A. Nikolovl, K. Pagem, A. Passantf, A. Shethg, K.Taylora, “The SSN Ontology of the W3C Semantic Sensor Network Incubator Group”, 2012

[20] M. Comptona, P. Barnaghib, L. Bermudezc, R. Garc´ıa-Castrod, O. Corchod, S. Coxe, J. Graybeal,M. Hauswirthf, C. Hensong, A. Herzogh, V. Huangi, K. Janowiczj, W. David Kelsey, Danh Le Phuocf,L. Leforta, M. Leggierif, H. Neuhaus, A. Nikolovl, K. Pagem, A. Passantf, A. Shethg, K.Taylora, “The SSN Ontology of the W3C Semantic Sensor Network Incubator Group”, 2012

[21] Michael Compton, "CSIRO Sensor Ontology", http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#CSIRO_Sensor_Ontology
[22] Danh Le Phuoc, "OntoSensor", http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#OntoSensor, 2008
[23] V.Tsiatsis, A. Gluhak, T. Bauge, F. Montagut, J. Bernat, M. Bauer, C. Villalonga, P. Barnaghi, S. Krco, The SENSEI Real World Internet Architecture, Ericsson Research, Ericsson AB, University of Surrey UK, Thales Research and Technology, SAP AG, Switzerland, Telefonica I+D, NEC Europe Ltd., ETH Zurich
[24] *Kevin Page, Definition of "SEEK OBOE",* http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#SEEK_OBOE
[25] *Kevin Page, Definition of "SEEK OBOE",* http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#SEEK_OBOE
[26] *Kevin Page, Definition of "SEEK OBOE",* http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#SEEK_OBOE
[27] Krzysztof Janowicz, definition of "Stimuli-Centered", http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#Stimuli-Centered
[28] Krzysztof Janowicz, definition of "Stimuli-Centered", http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/#Stimuli-Centered
[29] The Haskell Programming Language, http://www.haskell.org/haskellwiki/Haskell
[30] The Long Term Ecological Network, http://www.lternet.edu/
[31] Definition of "AlterNet", http://alternet.io/what-is-alternet/
[32] Mike Burrows, The Chubby lock service for loosely-coupled distributed systems, s, Google Inc., http://static.googleusercontent.com/media/research.google.com/el//archive/chubby-osdi06.pdf
[33] Thomas Fuhrmann, Pengfei Di and Kendy Kutzner, "Providing KBR Service for Multiple Applications", Universit¨at Karlsruhe (TH), Germany, TU Munich, Germany
[34] Definition of "ZooKeepr", https://cwiki.apache.org/confluence/display/ZOOKEEPER/Index, 2012
[35] Definition of HDFS, "HDFS Architecture Guide", http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[36] Atul Adya, Gregory Cooper, Daniel Myers, Michael Piatek, Thialfi: A Client Notification Service for Internet-Scale Applications, 2011