SEVENTH FRAMEWORK PROGRAME

Specific Targeted Research Project

Call Identifier:	FP7–ICT–2011–7–287305
Project Number:	287305
Project Acronym:	OpenIoT
Project Title:	Open source blueprint for large scale self-organizing cloud environments for IoT applications

D4.2.2 Utility Metrics Specifications

Document Id:	OpenIoT-D422-140114-V26
File Name:	OpenIoT-D422-140114-V26.doc
Document reference:	Deliverable 4.2.2
Version :	V26
Editor (s):	Jean-Paul Calbimonte
Organisation :	EPFL
Date :	2014 / 01 / 14
Document type:	Deliverable
Security:	PU (Public)

DOCUMENT HISTORY

Rev.	Author(s)	Organisation(s)	Date	Comments
V21	Jean-Paul Calbimonte	EPFL	2013/12/16	Initial version from 4.2.1
V22	Mehdi Riahi	EPFL	2013/12/23	Added content to Section 5. Adapted utility functions.
V23	Jean-Paul Calbimonte	EPFL	2014/01/03	Clarified relationship of main outcomes and goals. Revisions.
V24	Nikos Kefalakis, John Soldatos	AIT	2014/01/13	Added Chapter 7 on billing implementation, TR of the document (provided comments)
V25	Jean-Paul Calbimonte	EPFL	2014/01/14	Corrections following TR
V26	Arkady Zaslavsky	CSIRO	2014/01/14	Comments and Additions to Use cases.

TABLE OF CONTENTS

TERMS AND ACRONYMS **4**

1 INTRODUCTION **5**

1.1 Scope 5
1.2 Audience 5
1.3 Summary 5
1.4 Structure 6

2 UTILITY METRICS FOR PHYSICAL SENSORS AND ICO **8**

3 UTILITY METRICS FOR A SENSOR NETWORK AND APPLICATION SERVICE LEVEL **10**

4 UTILITY METRICS FOR VIRTUAL SENSORS AND ICO **12**

4.1 Virtual Sensors in OpenIoT 12
4.2 Parameters for defining and calculating virtual-sensor utility 12
4.3 Virtual Sensor Parameters for Accounting and Billing 13

5 UTILITY FUNCTIONS FOR VIRTUAL SENSORS AND ICO **16**

5.1 Review of Utility-based Metrics and Functions 16
5.1.1 Community Sensing Utility Functions *16*
5.1.2 Privacy in Community Sensing Utility Functions *17*
5.1.3 Bargain-based Utility Functions *18*
5.2 Utility Functions for Efficient Data Acquisition in OpenIoT 18
5.2.1 Utility-based Optimizer in OpenIoT *19*
5.2.2 Sensor Sharing in OpenIoT *21*
5.3 Utility Functions in Selected Query Types 22

6 UTILITY METRICS FOR OPENIOT USE CASES **25**

6.1 Utility Metrics for the OpenIoT Manufacturing Use Case 25
6.1.1 Overview *25*
6.1.2 Utility Calculation Model *26*
6.2 Utility Metrics for the OpenIoT Smart Cities Use Cases 27
6.3 Utility Metrics for the OpenIoT Digital Agriculture Phenonet Use Case 29

7 IOT BILLING BASED ON OPENIOT UTILITY METRICS **32**

7.1 Overview 32
7.2 Data Structures 32
7.2.1 User information data *32*
7.2.2 OpenIoT usage data *33*
7.2.3 Customer invoice and billing data *35*
7.3 Utility Metering Process 38
7.4 Scenarios Validation 39

8 CONCLUSIONS **41**

5 REFERENCES **43**

LIST OF FIGURES

FIGURE 1: GENERAL UTILITY-BASED OPTIMIZATION FOR PARTICIPATORY SENSING. 19
FIGURE 2: MYOPIC APPROACH: FOCUSING ON MONITORING A HOTSPOT, AT EACH TIME SLOT. 20
FIGURE 3: SENSOR SHARING BETWEEN DIFFERENT TYPES OF SIMULTANEOUS QUERIES. 21
FIGURE 4: TYPES OF QUERIES: A CLASSIFICATION 22
FIGURE 5: LOCATION MONITORING QUERIES. 23
FIGURE 6: REGION MONITORING QUERY. 23
FIGURE 7: APPROACH FOR A QUERY MIX SENSOR DATA ACQUISITION. 24
FIGURE 8. USE OF VIRTUAL SENSORS IN THE MANUFACTURING USE CASE (AND THEIR ASSOCIATED TO PHYSICAL DEVICES) 26
FIGURE 9: SILVER ANGEL SMART MEETING SCENARIO. 28
FIGURE 10: PHENONET ARCHITECTURE BASED ON OPENIOT PLATFORM 30
FIGURE 12: OPENIOT USAGE/UTILITY DATA 35
FIGURE 12: INVOICE AND BILLING DATA OF THE OPENIOT BILLING FRAMEWORK 36
FIGURE 13 CUSTOMER PAYMENT DATA STRUCTURES 37
FIGURE 14 CUSTOMER INVOICE AND BILLING DATA SCHEMA 38
FIGURE 15 APP USAGE REPORT OBJECT 39
FIGURE 16 BILLING SCHEME ASSIGNMENT CLASS STRUCTURE 40

TERMS AND ACRONYMS

DoW	Description-of-Work
EPC	Electronic Product Code
FI	Future Internet
FIA	Future Internet Assembly
GSN	Global Sensor Networks
HTTP	Hypertext Transfer Protocol
ICO	Internet-Connected Objects
ICT	Information and Communication Technologies
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IERC	Research Cluster for the Internet of Things
IoT	Internet of Things
IP	Internet Protocol
ITK	Industrial Traceability Kiosk
LSM	Linked Sensor Middleware
P2P	Peer-to-Peer
RFID	Radio Frequency Identification
SME	Small Medium Enterprise
SSN	Semantic Sensor Networks
W3C	World Wide Web Consortium
VO	Virtual Object
VS	Virtual Sensor
WSN	Wireless Sensor Network
XGSN	Extended GSN

1 INTRODUCTION

1.1 Scope

The purpose of this deliverable is to present a set of utility metrics that will be used in the scope of the OpenIoT utility computing paradigm. This answers to one of the goals of **WP4**, related to research and devise mechanisms for the autonomic and automated formulation of societies of internet-connected objects in response to **requests for utility services**. As part of this paradigm, a variety of utility-based algorithms will be designed and deployed, notably regarding resource management, utility-driven privacy and utility-driven optimization mechanisms. These algorithms have been **designed and implemented as part of WP5** of the OpenIoT project, yet they **make use of the utility metrics** outlined in the scope of this document. In addition to resource management, optimization, privacy and security, utility metrics will serve as a basis for accounting and management of SLAs (Service Level Agreements) between the OpenIoT cloud services providers and end-users. The present document constitutes the final version of the OpenIoT utility metrics specifications, along with some examples of using these utility metrics in different contexts. In particular, the present (and final) version of this deliverable presents how the utility metrics and the utility manager of the OpenIoT architecture can be used to drive utility-based privacy and a tangible billing implementation.

1.2 Audience

This document is addressed to researchers, engineers and other IoT experts both within and outside the OpenIoT consortium. Within the OpenIoT consortium this document will be consulted by the implementers of the OpenIoT middleware platform, as well as by researchers that work on the utility-driven mechanisms and algorithms of the project (notably researchers working in WP5 of the project). The former (implementers) need to consult this document as part of the implementation of the Utility Manager component of the project, which will keep track of all the utility-related parameters. At the same time, the latter (researchers) will have to take into account this document as part of their efforts to design and implement utility driven mechanisms for resource management and the overall efficiency of the OpenIoT infrastructure.

Outside OpenIoT consortium's members, IoT experts wishing to gain insights on utility calculation for both in terms of both metrics and algorithms also can consult this document. In the scope of IoT applications the topic of utility driven IoT metrics is still in its infancy and therefore the present deliverable is one of the first efforts towards addressing metrics for utility-driven services in the IoT domain.

1.3 Summary

This deliverable specifies the utility metrics that are considered and used in the scope of the OpenIoT project. These utility metrics are recorded as part of the implementation of the Utility Manager component of the OpenIoT platform, while they have also been used to **drive the utility based optimization mechanisms** of the project in WP5. This is strictly related to the OpenIoT task **T4.3 Utility Metrics Specification**. In particular we provided the following contributions:

- We provide an analysis and **summary of utility metrics** for different data providers and environments, **including physical sensors, sensor networks, and virtual sensors**. These metrics can be used to measure utility for interconnected objects.
- We proposed **utility functions** that use metrics in order to compute valuation and cost functions. These functions can be used by **utility-based optimization** techniques (implemented as part of **WP5, for task T5.1**). The utility based schemes proposed provide means and algorithms that can help selecting virtual sensors for **efficient data collection**.
- We describe **utility metrics**, tailored specifically for the **OpenIoT use cases**, indicating the relevant parameters (e.g. location, bandwidth, availability, privacy), and cost and valuation functions (if applicable).

We explore how these metrics can be used to optimize data acquisition, **as indicated in Task T4.3, including type of ICOs, data quality, data transmitted, bandwidth, location of ICOs, etc.** It is clear that for different use cases, the metrics can be more or less relevant, and we provide details about these specificities at the end of this document.

The deliverable classifies the various utility metrics into two broad categories, namely utility metrics for physical sensors (such as energy, bandwidth, data volumes) and metrics for virtual sensors (which in several cases coincide with those of the physical sensors). We report in this document how these metrics can be used in order to formulate utility functions that take into account the heterogeneity of queries that can be posed to the OpenIoT platform. Queries from users and applications can potentially share many sensors among them, and using all the data from all of them at the same time is not efficient. For this, we propose utility functions that can help optimizing the data acquisition process, and which have been used in WP5.

Furthermore, these metrics are related to the OpenIoT use cases, notably the use cases for which location, metering, accounting and billing makes more sense. Hence, specific examples are given on how these metrics can and will be used for metering in the scope of the OpenIoT use cases.

In addition to specifying metrics for utility driven functions, the deliverable reviews also popular algorithms and schemes for metering and accounting, notably schemes inspired from internet networking. These schemes are transferred and described in the IoT domain. They include flat-rate schemes, time-based schemes, volume-based schemes, smart-market schemes and more. Most of them are applicable to several IoT applications and hence could be implemented as add-ons to the OpenIoT platform. The implementation of all these schemes is however out of the scope of the OpenIoT workplan, yet they can serve as a sound basis for contributions by the open source community.

1.4 Structure

This deliverable is structured as follows:

- Section 2 (following this introductory section), provides an overview of utility metrics associated with physical sensors, notably metrics that are commonly used in the scope of sensor based applications.

- Section 3 focuses on utility metrics for sensor networks and related application services (i.e. it also focuses on metrics associated with the deployment and delivery of sensor network applications).
- Section 4 presents metrics associated with the utility of virtual sensors. It also illustrates possible algorithms for metering, accounting and billing, based on the use of the proposed/introduced metrics.
- Section 5 describes how the utility functions can be used to drive utility-based privacy implementations in the context of OpenIoT.
- Section 6 is focused on the specification of utility metrics associated with the OpenIoT use cases, notably in terms of manufacturing where examples of utility calculation are also given.
- Section 7 illustrates the use of the utility metrics and the utility manager of the OpenIoT architecture in order to support billing functionalities for IoT.
- Section 8 is the concluding section of this deliverable.

2 UTILITY METRICS FOR PHYSICAL SENSORS AND ICO

The physical sensors and ICOs are the fundamental (lowest level) data producers in OpenIoT. We can distinguish the following most important utility metrics for physical sensors and ICOs.

1. **Quality:** The quality of sensors and ICOs is the most fundamental metric that determines the accuracy and sensitivity of the measurements provided by a sensor and it may also influence energy consumption. Thus, in a given SN (Sensor Network) we may have a mix of high quality expensive sensors and low quality inexpensive sensors.
2. **Energy consumption**: Energy consumption is one of the most crucial utility metrics for sensor systems and more specifically wireless sensor networks (WSN). In WSN energy consumption is directly associated with the lifetime of the sensor network and therefore this metric can be used for functions like accounting, resource optimization and billing. The use of energy consumption in order to measure utility requires the introduction of an appropriate energy model [Schmidt 2007], which is usually a very complex research problem. Based on various energy models, the consumption is usually simulated [Shnayder 2004], yet in several cases they are also used for on-line accounting or resource management [Dunkels 2007] (e.g., in software). However, most of the energy models are usually simply taking for example into account the (total) number/volume of packets/data sent. Even though the volume of data is analogous to the energy consumption, other factors (such as the energy spent when listening for packets and the energy consumption of sensors' microcontrollers) should be also taken into account. In some cases these factors lead to more energy consumptions than the data transmitted though the network. To the extent that OpenIoT will rely on the measurement of WSN energy as a utility metric, the project's platform shall integrate one or more energy models. There are two options for the integration of these models within the OpenIoT architecture: (a) Integration of a single set of global models for WSN energy consumption, which should become part of the utility manager of the project. Each WSN could then be associated with one or more of these models, as part of its announcement to the OpenIoT directory service and (b) Integration of WSN specific energy models, falling in the responsibility of the WSN owner/provider. In this case the energy models should be integrated with the GSN nodes, which will undertake to communicate these models or the (resulting) utility metrics to the utility manager of the project.
3. **Bandwidth**: The bandwidth of a physical sensor refers to a bit-rate measure, representing the available or consumed data communication resources expressed in bits per second or multiples of it (bit/s, kbit/s, Mbit/s, Gbit/s, etc.). Note that in signal processing the word 'bandwidth' is used to refer to analog signal bandwidth measured in hertz. The connection is that according to Hartley's, the digital data rate limit (or channel capacity) of a physical communication link is proportional to its bandwidth in hertz. Thus, the utility of a sensor can be measured as an available or consumed bandwidth in the scope of an application.
4. **Data volume**: The volume of data (amount of data) produced by a physical sensor can be used as a utility metric. The more data streamed by the sensor, the more the utilization of the sensor. Hence, the utility of the sensor can be

analogous to the volume of sensor data streamed or consumed in the scope of an application. Thus, the utility of a sensor can be measured as a delivered data volume in the scope of an application.

5. **Trustworthiness**: The trustworthiness of a sensor can be measured as a trust one can place on a sensor that it will deliver true measurements on time within the scope of its technical parameters. Thus, the trustworthiness is related to the quality of a sensor.

For each physical sensor registered in the OpenIoT sensor directory, the OpenIoT utility manager will keep track of the following five utility parameters i.e. quality (as a semi-static value), energy consumption, bandwidth (in the scope of an application or service or time window), data volume (in the scope of an application or service or time window), and trustworthiness (as a semi-static value correlated with the quality of the sensor).

3 UTILITY METRICS FOR A SENSOR NETWORK AND APPLICATION SERVICE LEVEL

Most IoT applications and services comprise a large number of connected sensors. Therefore at the SN and application service level, the utility metrics combine utility metrics related to the sensors and the network itself. Thus, the utility of SN and the application service level can be measured based on the following parameters:

1. **System Lifetime**: System lifetime determines the longevity of the nodes. It is the most crucial factor in SN implying that the energy has to be utilized in the most effective way that is possible.
2. **Latency**: Latency measures the time delay experienced in a system where the lower limit of latency is determined by the medium being used for communications. In reliable communication systems, latency limits the maximum rate that information can be transmitted, as there is often a limit on the amount of information that is "in-flight" at any one moment.
3. **Quality**: Quality of SN is determined by the quality of data provided in response to a query.
4. **Delay** and **Delay Variation:** The delay and delay variation refer to delay in data collection from nodes.
5. **Bandwidth**, **Capacity** and **Throughput**: These indicate the capacity of data which can be sent over a link within a given time.
6. **Hop Count**: Hop count in communication determines the cost of a path, and eventually the energy consumed in the process.
7. **Ease of Deployment:** SN nodes have to be able to communicate with each other even in absence of established network infrastructure.
8. **Reliability**: Reliability measures the ability of the system or components to perform its required functions under stated conditions for a specified period of time.
9. **Survivability**: Survivability is the quantified ability of the system subsystem to continue to function during and after a natural or man-made disturbance. In particular, a survivable system should have **self-healing** mechanisms to cope with data transfer corruptions and failed nodes.
10. **Scalability**: Scalability measures the ability of the system to accommodate growth in the number of nodes such that the performance of the system does not deteriorate to the point where the system is unusable.
11. **Resource optimization and cost efficiency**: Resource optimization and cost efficiency measures the ability of the system to maximize the social welfare defined as an efficient allocation of limited resources in a society to optimize the resource utilization. It is well known that a system is efficient if and only if the system's social welfare is maximized. Thus, the social welfare of a system is equal to the difference between the system benefit and the system cost.
12. **Relevance**: Given an information need of a user of a WSN that is expressed by a corresponding query, the utility of a given sensor can be measured by its usefulness in answering the query. For example, given a query asking for CO2 the highest utility will have sensors providing CO2 while sensors providing

temperature will have lower utility since by receiving temperature one can obtain an approximate value of CO2 by exploiting the dependence between CO2 concentration and temperature.

13. **Confidentiality**: Confidentiality refers to the ability of the system to protect privacy of results provided for a particular user such that multiple users are not afraid that some aspect of their business can be compromised.

The above metrics will be also considered for any SN to be used and integrated within the OpenIoT platform and the OpenIoT sensor directory. Note that most of the above parameters map to static or semi-static values that can later used for composite cost calculations. However, OpenIoT will be also integrating virtual sensors, with relevant utility metrics as described in the following paragraph.

4 UTILITY METRICS FOR VIRTUAL SENSORS AND ICO

4.1 Virtual Sensors in OpenIoT

The previous section has underlined utility metrics that can be used for accounting with physical sensors in the scope of the OpenIoT platform. It is however the case that several of the sensors that will be integrated with the OpenIoT sensor cloud infrastructure will be virtual sensors rather than physical sensors. This is also the case since most of the sensors are likely provided by third-party providers of sensing infrastructures (instead of the OpenIoT cloud service provider) and integrated through the GSN middleware. Therefore, OpenIoT will in several cases have to deal with virtual sensors that announce themselves to the semantic directory service, and accordingly stream their data to the cloud (database). Furthermore, it is likely that OpenIoT will have to deal with virtual sensors, without knowing or controlling the composition of the physical sensors that contribute to the production and streaming of the virtual sensor data. For this reason, OpenIoT will have to keep track of the utility of virtual sensors, which bear several differences associated from physical sensors. For example, while energy models and metrics are important for physical wireless sensor networks, they cannot be directly associated with virtual sensors since the information about which physical sensors comprise the virtual sensors might not be generally available to the OpenIoT platform. On the other hand, there are also utility metrics that are common to both physical devices and virtual sensors, such as the volume of the transmitted data and the bandwidth consumed. The following paragraph attempts a discussion of utility metrics that are applicable to virtual sensors.

4.2 Parameters for defining and calculating virtual-sensor utility

The following parameters can be used to measure the utility of virtual sensors:

1. **Data Volume**: The volume of data (i.e. number of bytes) streamed by the virtual sensors can be used as a utility metric. The more data streamed by the virtual sensor, the more the utilization of the virtual sensor. Hence, the utility of the sensor can be analogous to the volume of sensor data streamed or consumed in the scope of an application.
2. **Bandwidth**: Directly related to the data volume of a virtual sensor is the bandwidth (i.e. bytes/sec) consumed/associated with the data volume streamed by the virtual sensor i.e. the rate of data streaming. Similarly to the previous case the utility of a sensor can be analogous to the bandwidth consumed (by the virtual sensor) in the scope of an application.
3. **Time of the Usage Session:** Virtual sensors can be used in the scope of application sessions. The time during which a sensor has been used can serve as a metric for utility calculation. Note however that there are different ways to define the timing boundaries associated with the usage of a sensor (e.g., according to the overall application session where the virtual sensor is used, or the actual time a specific virtual sensor has been occupied). In order to keep track of the time boundaries the start and finish time associated with the use of a resource or a service should be recorded.

4. **Virtual Sensor Location**: The location of a virtual sensor is another prominent parameter that defines its value (or utility). Different locations can signify different business values for the same sensor.
5. **Virtual Sensor Task**: In the scope of business processes, virtual sensors serve some task (process step). The relative business value of this step can drive the definition of the utility or the business value of a sensor. Note that the task associated with a virtual sensor can in several cases be related with the location of the sensor.
6. **Number and type of Physical Sensors used**: A virtual sensor utility metric can be defined and calculated on the basis of the number and types of the physical sensors that comprise the virtual sensor. In particular, a weighted formula can be used to define and calculate the utility of the virtual sensors on the basis of one or more utility metrics associated with the physical sensors that comprise the virtual sensor. In this way, the utility metrics associated with physical sensors (e.g., energy consumption) (as defined in previous sections) could be used in order to calculate the utility of the virtual sensor.
7. **User Defined Cost**: Similar to the case of a physical sensor, a user-defined cost/utility value could be assigned to the sensor. The assignment of the user-defined cost could take into account the above criteria and parameters, but also other criteria defined by the owner, deployer or integrator of the sensors infrastructure.

The OpenIoT platform should keep track of the above parameters as a means to enable utility calculation for accounting, billing and resource management purposes. Resource management concerns primarily the service provider's perspective, while billing and accounting concerns also the end-user's perspective. For the latter, the utility should be ultimately assigned to the OpenIoT service, typically on the basis of a combination of the utility metrics for virtual sensors.

4.3 Virtual Sensor Parameters for Accounting and Billing

OpenIoT specifies and implements (as part of WP5) a set of utilitarian mechanisms for resource management, as well as utility-based privacy and security. Apart from providing support for these mechanisms, the recording of utility metrics (such as those outlined above), can enable the implementation of accounting and billing mechanisms as required by the utility nature of the OpenIoT sensor-cloud. In terms of billing mechanisms, a number of different schemes can be adopted based on approaches that have been proposed in literature, notably in the area of charging for Internet resources and services (e.g., see [Falkner 2000] and [Choi 2007] for a survey of main schemes and issues). Note that such schemes consider one of more virtual sensors comprising a service, as well as one or more of the virtual sensor utility metrics outlined above.

A summary of these approaches are provided below:

- **Flat-rate schemes**: Flat-rate schemes are the simplest billing schemes and are calculated on the basis of fixed tariffs for a specified amount of time. Flat-rate schemes should be based on the assignment of a utility-rate to OpenIoT services, which are typically provided based on the combination of multiple sensors (as already explained in deliverable D4.1 of this work package).

- **Time-based schemes**: On the basis of these schemes pricing is based on how long a service is used. In general time-based pricing bills for resources and services on the basis of the time a service or resource is utilized. The usage time associated with a resource or a service should be therefore taken into account. The price can be defined as a function of this time, and more specifically of the start and finish time. This is directly related to the time usage metric outlined in the previous section.
- **Volume-based schemes**: Volume based billing/pricing schemes apply functions over the volume of data incurred in the usage of the service. OpenIoT services will typically comprise multiple virtual (and physical sensors) and hence a volume based scheme will exploit the volume-related utility metrics of multiple sensors.
- **SLA (Service Level Agreement) or QoS (Quality of Service) based schemes**: Pricing is usually based on the quality of the service or the service level agreement associated with the utility services. These can be defined based on one or more of the defined utility metrics, including specific types of sensors, specific locations of the sensors, guaranteed volume of the sensors and more.
- **Priority-based schemes**: Priority schemes have their origins in the Internet, where services can be labelled and priced according to their priority. In this respect, priority schemes are relevant to the SLA/QoS based schemes. In OpenIoT, services can be labelled according to the number and type of sensors that they use, thereby getting some priority over others. Another alternative could be based on the SLA between end-users and OpenIoT service provider.
- **Schemes based on number, type and location of ICOs**: In the scope of internet-based pricing, there have been proposed schemes that calculate bills on the basis of the distance (or number of hops) between the service and the user. As a variation of this scheme, pricing could be based on the number, type and location of ICOs.
- **Effective volume / Session-oriented pricing schemes**: Session-oriented schemes are based on the calculation of an effective utilization of the resource entailed in a session. The calculation can take into account the anticipated time usages of the session, as well as the data volumes of the various sensors to be consumed. The prediction of the volume of data to be used gives rise to the characterization of these schemes as «effective volume» ones, based on the terms effective bandwidth that has its roots in internet traffic management research, yet it has also been used for wireless networks [Tse 1998].
- **Smart market based schemes**: Such schemes foresee pricing on the basis of an auction for specific resources (i.e. sensors / ICO) or services. They can be implemented on the basis of a dynamic market based regulation of the user-defined cost parameters associated with the virtual sensors outlined above. The idea is the more the demand (i.e. number of users/services asking for a sensor) the higher its price (i.e. user-defined cost/utility).

In addition to these schemes, several others can be proposed/derived on the basis of variations and combinations of the above, such as location-based schemes (i.e. charging according to the locations of the sensors) and content based pricing (i.e. pricing based on the type and volume of the content delivered to the user. Moreover, there are several classifications of the above schemes e.g., as static pricing schemes

(i.e. when the pricing function does not change over time) and as dynamic pricing schemes (i.e. when the pricing function is dynamically adapted to the behavior of the users).

The schemes outlined above can be used in order to combine the utility of the various sensors into utility metrics for wider applications or services that comprise multiple sensors. For example, a volume based scheme can be applied towards calculating the utility of an IoT service as a weighted sum of the utility metrics of the various sensors comprising the service.

Research and implementation of specific pricing schemes in not in the scope of OpenIoT research and development, However, as an IoT middleware framework, the OpenIoT platform will record and document the utility metrics needed in order to support the implementation of one or more of the above schemes. Hence, each integrator or solution provider using the OpenIoT open source software will be able to select and implement the above schemes. This is also a task that could be offered to interested members of the OpenIoT open source community.

5 UTILITY FUNCTIONS FOR VIRTUAL SENSORS AND ICO

Up to now we have presented utility metrics applicable for different types of data sources including physical sensors, sensor networks and virtual sensors. These are of particular interest for the OpenIoT platform, as they represent the underlying ICOs whose data is provided and queried across the web.

In this section we dive deeper in the details of how these metrics can be used in order to propose specific utility functions for an IoT environment. The utility functions, in general, need to be maximized, representing an increase in the social welfare. Nevertheless, defining such functions is not a simple task, as there are many variables and metrics, and these also depend on different costs and valuation functions that might need to be defined.

We present a review of utility functions that have been used in the literature, taking into account participatory sensing, community driven utility functions, privacy concerns, equilibrium in an IoT society and stimulation mechanisms. Then, we propose an approach for defining utility functions for OpenIoT, considering different types of queries or requests from heterogeneous users, and taking into account utility metrics defined previously. The analysis and results of this work have been used to create optimization mechanisms and algorithms for data acquisition, which are described in OpenIoT Deliverable D5.1.2. This section has been partially adapted from [Riahi 2013].

5.1 Review of Utility-based Metrics and Functions

Several approaches exist in the literature for defining utility functions for sensor data collection, acquisition, billing and service delivery, optimization, etc. In this section we review some of these, focusing mainly on community and participatory sensing. This is a particularly interesting setting, as it has to deal with the heterogeneity of the ICOs and virtual sensors that provide data to the cloud. This is precisely the type of environment that OpenIoT use-cases are related to. Data provided by ICOs can be queried by users and applications through very different schemes and with very different purposes. Moreover, privacy can be an additional cost in some scenarios while in others (e.g. public weather information) these restrictions might be relaxed. Also, and as it is the case in the OpenIoT use-cases, location-based queries are also of particular importance, although several sensors may provide overlapping data whose value may decrease as the degree of overlap increases.

In this section we first analyse specific functions for community sensing. Then we present our proposal for utility-based functions, which are used in the OpenIoT prototypes presented in the Deliverable 5.1.2.

5.1.1 Community Sensing Utility Functions

One of the first attempts at establishing utility-based community sensing [Krause 2008], the approach consists in weighting information needs based on the expected demand by the users. In this setting, the application selects sensors from the available ones in such way that the sensor readings improve the most demanded aspects of the phenomenon model. This model can be formalized as a stochastic process within spatial boundaries (e.g. a random variable X_s for a location $s \in V$). The goal in this approach is to select a set A of locations such that the variance of

unobserved locations is minimized. Given a particular subset of locations $B \in W$ of all possible locations W, we introduce $P(A \mid B)$ as the distribution over subsets A. The objective, then is to maximize the following function:

$$F(B) = \mathbb{E}_{A|B}[R(A)] = \sum_{A} P(A|B)R(A)$$

Where $R(A)$ is the expected demand-weighted variance reduction over the location subset A. Then, in order to select a set of locations B, one needs to consider the inherent costs (denoted as a function $C(B)$), which should be limited by a budget L. This is the restriction for the final optimization problem, which has been shown to be NP-hard. Therefore, approximation algorithms have been proposed for solving it.

Some of the assumptions made for this approach are not really applicable for most of the OpenIoT scenarios. First, it does not consider the heterogeneity of applications and users, which may potentially launch several different types of queries, and not only requesting one type of phenomenon. Moreover, there is not any notion of trustworthiness or quality of the sensing participants. In a more generic scenario, each application might have its own utility and costs, and the obtained value depends on the quality of the received data.

5.1.2 Privacy in Community Sensing Utility Functions

Data from ICOs may inherently contain private information, even when it is aggregated within an IoT community. Users may find benefits when sharing part of this data (e.g. from social status, to services provided by the services of the IoT). However the users may want to preserve their privacy, at least to some level of granularity, balancing the costs and benefits of data sharing. Following these lines, game-theoretic approaches such as [Liu 2008] have been devised, in which users reveal their location, only at certain level of granularity. More formally, a user i from the set of all users N, can see only a subset $N(i)$ of nearby users. The granularity of the location of user i is given by some value $a_i \in [a_{min}, a_{max}]$. Then the following utility function is defined:

$$U(a_i, a_{-i}) = \min\left(K, \sum_{j \in N(i)} \min(a_i, a_j)\right) - c \cdot a_i$$

Where $\min(a_i, a_j)$ is the perceived accuracy of user j for user i. K is a predefined constant that indicates an upper bound of the user benefits. a_{-i} is a strategy vector for all other users except i ; and c is a penalty factor.

While this approach includes is rather simplistic, it introduces an interesting game-theory scheme that looks for optimality in an environment where all users look for maximizing their gain, but not in a greedy way. Moreover, the penalty factor in this utility function is under-specified, and should be more general than just a constant factor. In more realistic scenarios, privacy and resource consumption are parameters that must be included in cost factors. Finally, this utility-based approach assumes that all utility functions are homogeneous, while in OpenIoT we add the possibility of users having different privacy levels and thresholds.

5.1.3 Bargain-based Utility Functions

In the Internet of Things, when users or applications provide data to the cloud, they become participants of the overall system. These participants may be motivated by rewards that they could gain for providing their data. In the approach proposed by [Xie 2009], users are selfish and rational but not malicious, trying to maximize their rewards. All participating nodes communicate through messages to a sink node.

More formally, this formalization introduces the expected credit reward, defined as:

$$R_i^m(r) = A_i^m(r) \times P_i(r)$$

Where $P_i(r)$ is the contact probability of node i with the sink node of message r. $P_i(r)$ denotes the message appraisal associated with each message. It indicates the probability that no other node has provided a copy of the message to the sink. From this, the following utility function for a node i is defined:

$$S_i = \sum_{r=1}^{R} \left(\sum_{m \in \phi(r)} R_i^m(r) - \sum_{m \in \psi(r)} R_i^m(r) \right)$$

Where R is the total number of message types, $\phi(r)$ and $\psi(r)$ are messages of types r, after and before exchange. Participants exchange message lists, and start a bargaining process as a cooperative game. To find an optimal solution for this bargaining process [Xie 2009] proposes a Nash optimal solution through a greedy algorithm with polynomial time complexity.

Although this proposal introduces an interesting bargaining model that may have similarities with OpenIoT user cost-benefit expectations, it has some limitations. These include the negligible cost of exchanging control information, and the possible duplication of appraisal values.

5.2 Utility Functions for Efficient Data Acquisition in OpenIoT

There exist many possible usages of utility metrics in the context of OpenIoT. In the project, we have settled mainly on looking for optimization schemes that use these metrics in order to efficiently use the available resources. More specifically, we are interested in the optimal usage of ICOs and the data that they provide to the platform. Considering the potentially very large number of sensors that act as data providers, it may happen that at different points in time, the OpenIoT platform may need only a fraction of the data that the complete set of available ICOs supplies. A mismanagement of these resources would directly translate in over-utilization of data, causing unnecessary bandwidth usage and potentially higher response times, etc.

To alleviate this, in OpenIoT we propose using a utility-based approach that maximizes the social welfare, while users and applications can have different data requirements (through different types of queries), and utility consideration (through different utility functions). We illustrate this scheme in Figure 1 in a general way, where an aggregator can potentially acquire data from a set of sensors *S*. Users may issue queries to the aggregator, requesting for instance the average CO2 level in a certain location or in a broader area. The aggregator is also aware of the sensors' capabilities, including energy, location, etc. and constraints (e.g. privacy

requirements, energy and bandwidth thresholds), which can be used to efficiently schedule the queries in such a way that the global utility is maximized.

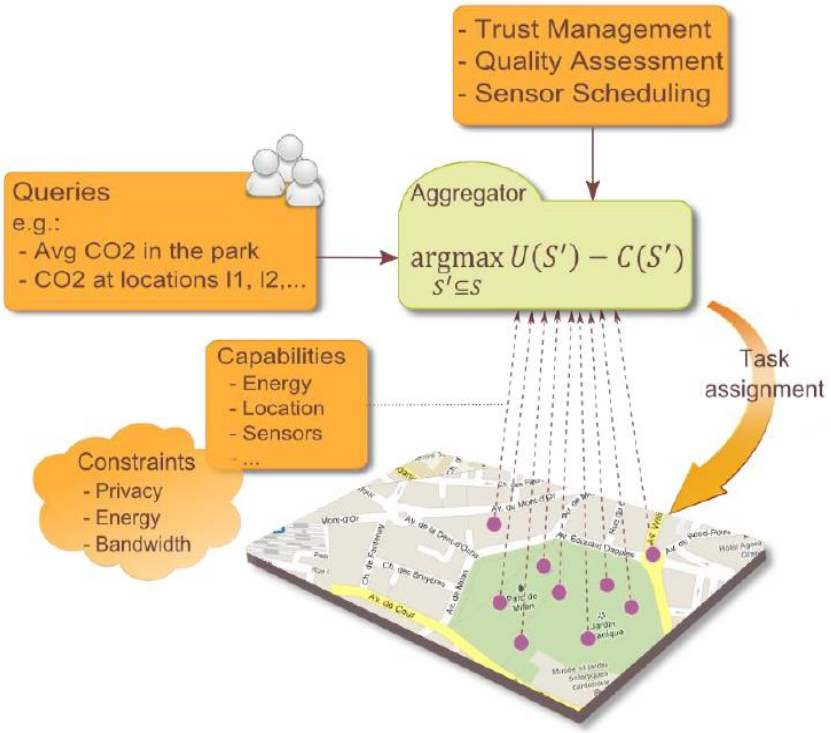

Figure 1: General utility-based optimization for participatory sensing.

This is a very general model that can be specialised for different use-cases and requirements. It also depends on the metrics that are being used, the utility functions that are chose, and also the types of queries that are received. The heterogeneity of queries make it highly complex to formulate an optimization scheme based on utility functions.

5.2.1 Utility-based Optimizer in OpenIoT

The proposed general utility-based model described above, has been specialized for the OpenIoT architecture. We propose a utility-based data acquisition approach, **which is explained in details in Deliverable 5.1.2** of the OpenIoT project. This approach is focused on a Utility-based optimizer that selects sensors, given the queries, budget and quality requirements from the OpenIoT Scheduler module. With this information, the optimizer should use the sensor metadata (e.g. including location, energy, privacy, etc.) to select a subset of sensors *S'* from the global set of sensors *S*, such that the valuation minus the cost of using this subset is maximal.

The optimizer, operating at the Local-scheduler level, is able to transform the original queries from the users, into rewritten queries that only use the subset *S'*, given as a result of the utility-based optimization. Then, based on the user needs X-GSN may activate or deactivate virtual sensors depending on the relevancy to the queries. While this **optimization techniques have been reported as part of WP5**, they use the **utility metrics, utility functions and cost schemes described henceforward, as part of WP4**.

This optimization can be formalized as follows. The objective is to acquire data for the queries from the available sensors in order to maximize the utility over a period T. We denote as Q the set of all queries up to T, $\mathcal{S}^t$ denotes the set of available sensors at time slot t, and $K: Q \rightarrow \times_{t=1}^{T} 2^{\mathcal{S}^t}$ defines an allocation scheme that assigns sensors to each query. $Y(K,t)$ is a function that returns the set of sensors that are assigned to all queries at time t. We represent the cost of sensor s at time t given the allocation K by $c_s(K,t)$. Let $\mathcal{K}$ denote the set of all possible allocation schemes. Then the goal is to find allocation $K^* \in \mathcal{K}$ that maximizes the *social welfare*:

$$K^* = \underset{K \in \mathcal{K}}{\operatorname{argmax}} \left(\sum_{q \in Q} v_q\big(K(q)\big) - \sum_{t=1}^{T} \sum_{s \in Y(K,t)} c_s(K,t) \right)$$

Tor solve this problem, we need to know in advance all the queries that will be issued over T, and the location and cost of all the sensors at each time slot. However, in the context of OpenIoT, users must be able to submit new queries whenever they desire and it is not realistic to ask the users to pose all their queries in the beginning of the period T, and their exact locations at a specific time slot cannot be determined a priori. Due to the lack of access to all the required information to solve the above long-term optimization problem, we resort to a *myopic* approach (Figure 2), in which we try to maximize the utility at the current time slot without considering the future state of the system. In this approach, when finding the optimal allocation scheme, we only consider the queries and sensors that are available at the current time slot. After finding the best allocation scheme, the cost of each selected sensor is shared among queries that are answered using the measurement from that sensor.

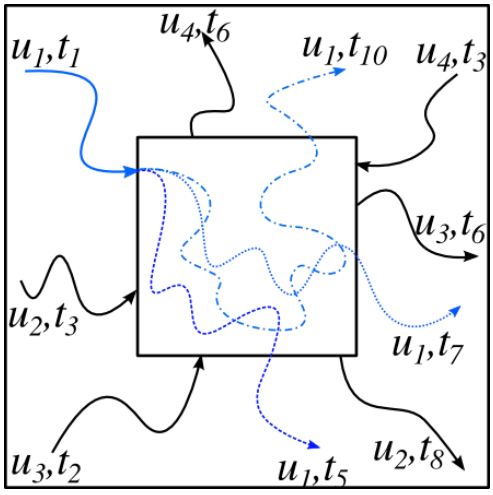

Figure 2: Myopic approach: focusing on monitoring a hotspot, at each time slot.

Formally the optimization can be defined as the allocation $M^* \in \mathcal{M}$ that maximizes the total utility in the current time slot:

$$M^* = \underset{M \in \mathcal{M}}{\operatorname{argmax}} \left(\sum_{q \in Q} v_q(M(q)) - \sum_{s \in Y(M)} c_s \right)$$

M is an allocation scheme that assigns sensors in S to each query q. $Y(M)$ is a function that returns the set of sensors assigned to queries.

Finally, we have to ensure that using the selected sensors S_q for a query q, the utility is positive:

$$v_q(S_q) - \sum_{s \in S_q} \pi_{q,s} > 0$$

Where $\pi_{q,s}$ is the price of q for using sensor s.

5.2.2 Sensor Sharing in OpenIoT

Now that we have defined the utility optimization problem, we need to consider how it can be solved. In general, this has been shown to be an NP-hard problem, and therefore we have proposed heuristic algorithms for different cases in [Riahi 2013]. Specifically, we focus in this section on different types of queries for which we have shown the feasibility of our approach. These types of queries emerge from real-world needs in an IoT environment where an arbitrary number of sensors in a location or an area can potentially be useful for an, also arbitrary, number of queries from users and applications. We illustrate this in Figure 3, where sensors (denoted as stars) can be shared among heterogeneous queries. A point query (in a particular location) might be answered by sensors nearby with different quality levels (higher or lower valuation). A trajectory query might require data form several sensors across its path. A Location monitoring query may need data from a specific location, but over a certain time frame, at different rates. In a more general region monitoring query, a number of sensors may overlap and provide different valuations at different costs.

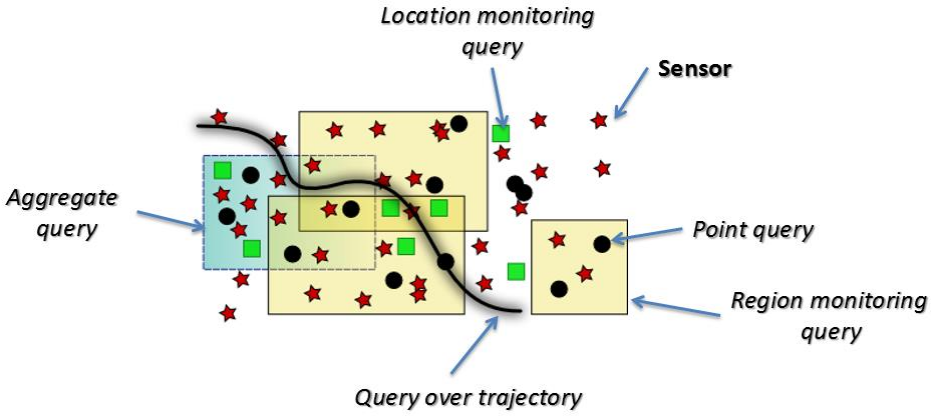

Figure 3: Sensor sharing between different types of simultaneous queries.

In our research, we have shown algorithms for some of these cases (Figure 4), in particular: **Point queries, spatial aggregate queries, Location and Region monitoring queries**.

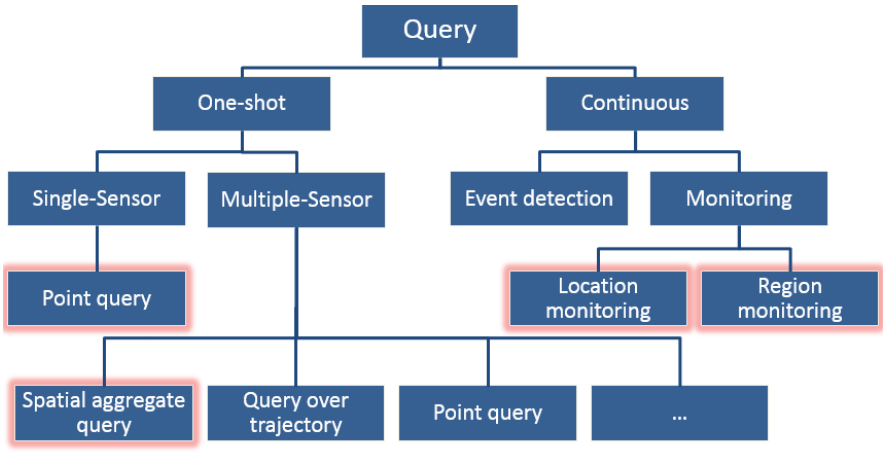

Figure 4: Types of queries: a classification

5.3 Utility Functions in Selected Query Types

For all cases, the valuation functions and costs have been detailed in Deliverable D5.1.2, where all these functions are used. Moreover the algorithms for each case and the experimental evaluation are fully detailed in [Riahi 2013]. In this section we mainly discuss the utility functions.

In single sensor point queries, users are interested in a phenomenon at certain location. For point one-shot queries, the utility function is defined as:

$$u(S') = \sum_{l \in L} \max_{s \in S'} v_l(s) - \sum_{s \in S'} c_s$$

Where S' is a subset of S, l is the location and v_l is the computed utility at location l. This is a non-monotone submodular function that has been to be solvable by an approximation LocalSearch algorithm.

For multi-sensor one-shot queries we require to select the set of sensors exploiting as much as possible the common data requirements among the posed queries. The proposed greedy algorithm for solving this optimization problem, iteratively selects sensors that maximize the partial overall utility. The utility function defined is given by:

$$u(S') = \sum_{q \in Q} v_q(S') - \sum_{s \in S'} c_s$$

Where again S' is the set of selected sensors and Q is the set of queries.

For the case of continuous queries (i.e. continuously executed during a time period), we can consider first a location monitoring query. In this case the user is interested at some phenomena at a given location, but over a certain time period (i.e. with a requested sampling time). However, the sensors may not always be able to provide

data at the same intervals (Figure 5). In such cases, if the budget allows, the data from other sampling times can be used for acquisition.

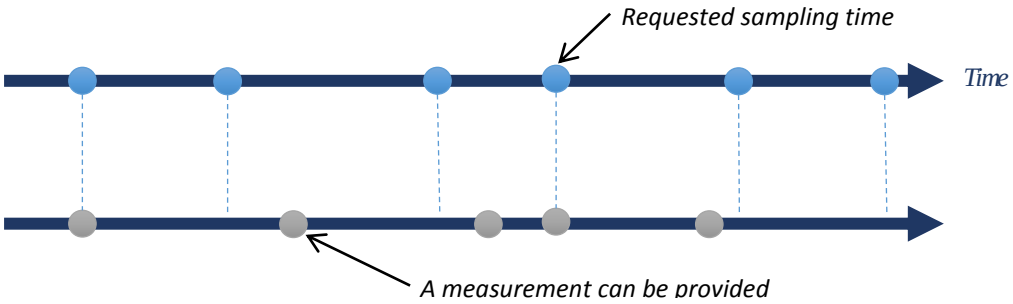

Figure 5: Location Monitoring Queries.

In summary, the problem of location monitoring queries can be solved by generating several point queries, answering those point queries using approximation algorithms as the one mentioned above, and then calculating the payments for each satisfied query. At the end of every time slot, the remaining budget can be updated for each query.

For region monitoring queries, sensor data sharing is possible if the regions over which the queries are executed overlap. A modified algorithm can take advantage of this, by providing a set of weighted costs of sensors. As an example, if a subset of sensors was already selected by another continuous query, then a weight of 0 can be assigned to that subset of sensors.

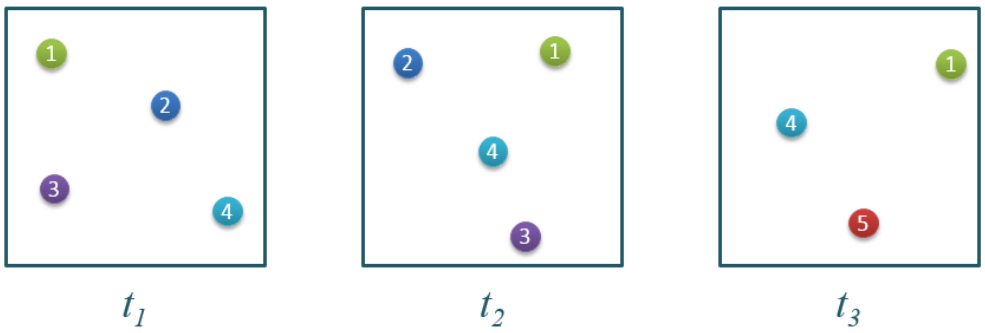

Figure 6: Region Monitoring Query.

As a final case, we can combine the abovementioned types of queries in a query mix, and the problem can be seen as a composition of the sub-cases described above. Figure 7 illustrates this case. The proposed algorithm for the query mix selects sensors by exploiting the commonalities of the queries posed to the system. It first generates point queries for location and region monitoring queries. Then, all queries are provided to the greedy algorithm used for multiple sensor one-shot queries, so that it optimizes the total utility. Afterwards, the results of the point queries are applied for continuous queries. As in this stage there might be queries sharing the same sensors (e.g. regions overlapping), the payments need to be adjusted accordingly.

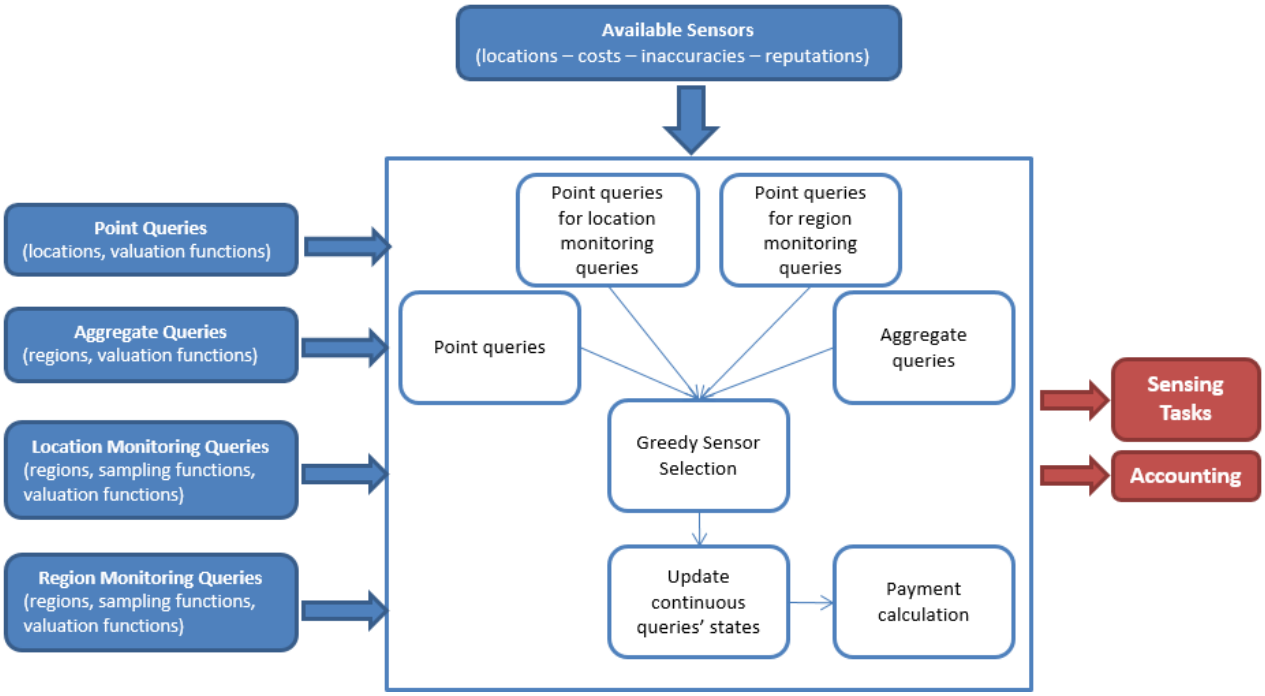

Figure 7: Approach for a Query Mix sensor data acquisition.

6 UTILITY METRICS FOR OPENIOT USE CASES

6.1 Utility Metrics for the OpenIoT Manufacturing Use Case

6.1.1 Overview

The manufacturing use case implemented by SENSAP in the scope of WP6 provides a good opportunity for implementing and using several of the utility metrics outlined above. As detailed in WP2 deliverables the use case concerns the exploitation of several physical sensors along with the OpenIoT framework in order to facilitate the on-demand selection and visualization of manufacturing KPIs for plant operators. The use case involves several physical sensors, which focus on the needs of the paper/packaging industry including:

- Color diffusion sensors, which are used to audit and make sure that all colors are printed correctly.
- Automated barcode scanners & camera-based verifiers (chiefly laser-triggered sensors).
- Weight sensors.
- Height sensors, mainly used to detect when containers are full.
- RFID sensors, usually as part of dock-door portals).

All the above sensors are integrated within the (Industrial Traceability Kiosk (ITK) product of SENSAP, which aggregates data from all the above physical sensors and sends them to SENSAP's S-BOX middleware product for further processing. As part of the use case implementation, S-BOX announces and integrates a number of virtual sensors to the OpenIoT platform. These virtual sensors correspond to manufacturing performance related events and are structured according to EPCglobal EPCIS standard (which provides the framework for placing these events in a given manufacturing context). Compliance to EPCglobal metadata and event mechanisms is accomplished through the S-BOX middleware, which undertakes to translate raw data streams to EPCglobal Application Level Events (ALE) and at a later stage to EPCIS streams.

Since virtual (rather than physical) sensors are integrated in the OpenIoT platform, utility calculation for the applications will leverage virtual sensor metrics, in-line with the general-purpose metrics listed in Section 3. In particular, the following metrics will be used to quantify the utility of the calculation of a manufacturing KPI as part of the application:

- The number of EPCIS stream processed in order to generate a virtual sensor result. In practice, this is proportional to the difficulty of estimating the requested (by the end-user (i.e. plant operator) KPI value. For example, the generation of a KPI for a logistics process (which typically has only phase) is much easier (and hence less costly) than generating a KPI for a manufacturing process (which typically spans multiple phases) need to install additional hardware/sensors to different locations to capture data)

- The number of raw data streams (physical sensor data) processed for generating EPCIS streams. The ability to keep track of this number stems directly from the recording of the physical sensors that are used by a virtual sensor.
- The event backlog/queue used by the virtual sensor. The queue/backlog of history events that is examined to produce a virtual sensor result. For example, in order to calculate the average production rate, an observation window of the last N hours of operation of the production is used. This event queue is a metric directly associated to the volume of data used for producing the virtual sensor result.
- The perceived added-value of information value for the organization (manufacturer), which can be mapped to percentage of the generated products' value). This added-value can be calculated on the basis of the user-defined value assigned to virtual sensors.

6.1.2 Utility Calculation Model

In-line with the general guidelines for utility calculation presented above, Figure 8 illustrates the process of combining physical sensor measurements from the manufacturing shop floor towards virtual sensors. As shown in the figure, the virtual sensors of the OpenIoT manufacturing use case are composed based on a series of sensor-events that comply to the EPCIS standard. A virtual sensor is typically based on a combination of real-time (on-line) events and historical events (also EPCS compliant) that are stored in an EPCIS repository.

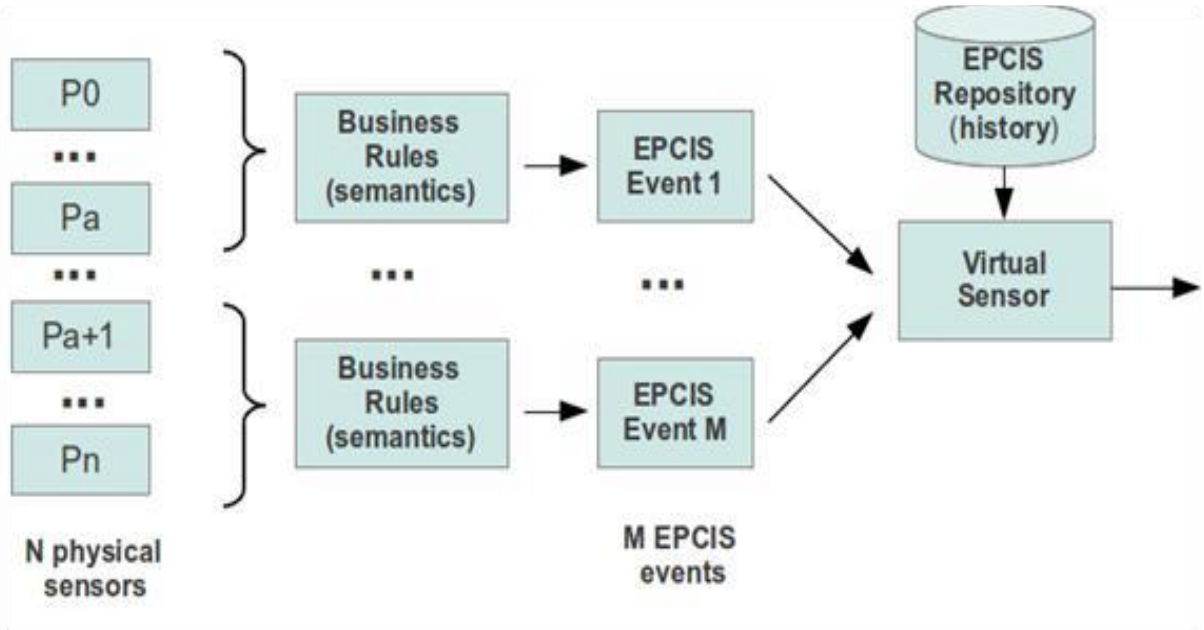

Figure 8. Use of Virtual Sensors in the Manufacturing Use Case (and their associated to physical devices)

The utility calculation model is therefore applied over virtual sensors according to the model depicted in Figure 8. In particular, for a given process P, n physical sensors are deployed. An a priori known operational cost CN_i is assigned to each physical sensor (on the basis of parameters such as power requirements, labour costs to it setup, monetary cost of the sensor etc).

Each physical sensor i produces a raw stream S_i that is in turn processed by a set of business rules yielding m EPCIS events. A data acquisition and filtering cost CB_j is assigned to each one of the M EPCIS events. This cost factors in the computational costs involved in multiplexing the physical sensor events and the application of business rules (which may need to perform multiple queries to external data sources).

If the virtual sensor needs to query history data from an external EPCIS repository we also need to apply a data retrieval cost $CD = g(w)$ where w is the size of the history window that we are retrieving. Note that the external EPCIS repository contains historic EPCIS events (values) derived previously.

Finally, the organization assigns an *importance* metric $I = f(P)$ which is a function of the monitored process and indicates the level of importance of the sensor output for the process execution. For example, early detection of printing errors at the beginning of a high-cost manufacturing process is very important to a company whereas knowing the exact number of produced items during the process execution is of less importance. This importance metrics is organization and business process specific.

Given the above model, the utilization of our setup can be estimated on the basis of the following formula:

$$U = \sum_{i=1}^{n} CN_i + \sum_{j=1}^{m} CB_j + a \bullet g(w) + f(P)$$

where $a = 0$ *if no EPCIS history data is required, 1 otherwise.*

6.2 Utility Metrics for the OpenIoT Smart Cities Use Cases

A large number of metrics can also be used for the different services that are usually deployed in the scope of smart cities (e.g., overall metrics for energy efficiency, volumes of data collected and more). In the more specific context of the Campus Guide use case there will be a need to keep track of the service utilization. In particular, applications like facility management services or accounting management will need to be aware of the actual usage of the service. To this end, service level metrics shall be recorded upon each invocation of the IoT services that comprise the use case.

In the Campus Guide application, users may start conversation and interchange messages and objects through their QR codes using their mobile devices. While the rooms and classes are rather static objects, and their data is not too dynamic, the users' locations are. Therefore it might be of interest to set up queries that monitor the presence of other students in the same location, or around the same room. Here, the previously described location monitoring queries could be applied to some extent.

Nevertheless, other simple considerations could be taken to define the valuation functions. For example, students not taking a course at the present time, have a utility value of zero as they are not potential participants in a discussion. In general the following metrics can be of use in this case:

- **Bandwidth**: the bandwidth (i.e. bytes/sec) consumed/associated with the data volume streamed by a virtual sensor i.e. the rate of data streaming.
- **Time of the Usage Session:** Virtual sensors can be used in the scope of application sessions. The usage of sensors in this use case might be directly associated to the conversation or discussion in the smart campus.
- **Virtual Sensor Location**: The location of a virtual sensor is a key parameter in this use case, considering that proximity between participants, e.g. in the boundaries of a room. ICOs in a distant location may have no value at all for a user.

For the Silver Angel use case, we can take as example the smart meeting scenario (Figure 9), as it considers many of the query requirements and privacy concerns discussed in Section 5.

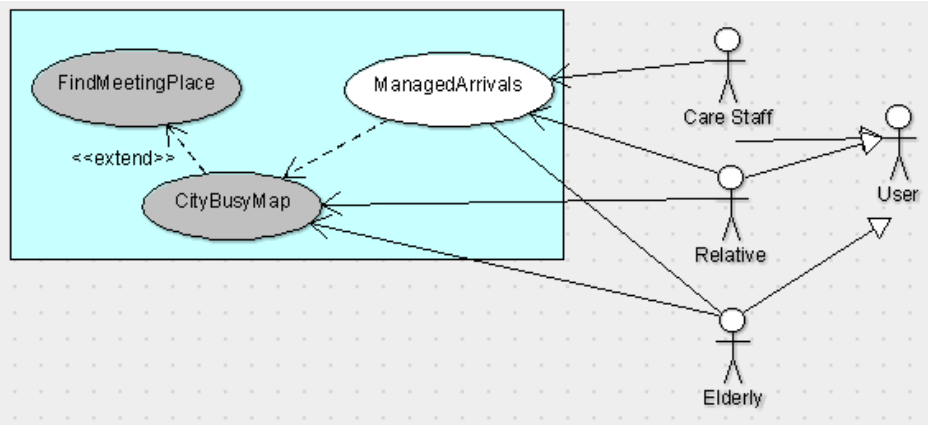

Figure 9: Silver Angel smart meeting scenario.

As it is described in Deliverable 6.3.1, in this scenario a meeting can be proposed by finding a place that meets certain criteria, which could include:

- People (few/many/ignored)
- Noise (low/high/ignored)
- Pollen levels (low or ignored)
- Air quality, etc.

In this case the community sensing approach we described can be a basis for defining the utility functions. Queries may include for instance requesting the current noise levels in a location, or the pollen levels in a limited region. According to these criteria, one or several sensors could be chosen, in such way that the overall benefit is obtained. More precisely we can enumerate some of the metrics to consider:

- **Location**: this is again a key parameter. Only those sensors in the location, region or boundaries of the meeting is relevant to the query and the value of other participant sensors in negligible in terms of utility.

- **Accuracy**: Depending on the accuracy of the sensor observations (e.g. accuracy of the air quality monitor) a particular sensor may have less value to the target query.
- **Trustworthiness**: The data for certain observations may be overlapping when two or more sensors provide it for the same area. However, sensors deployed by a trustworthy source may be weighted more than, for instance a home-made sensor provided by an anonymous citizen.
- **Privacy**: For instance a user providing noise levels using her smartphone will be concerned if the values provided allow in some way that her position is disclosed. Cost functions associated can be raised using the privacy as a parameter, in order to make it more expensive to get results from a particular participant in every time period.
- **Acquisition rate**: even if a sensor is able to provide the expected measurements at the requested location, the rate at which the data is gathered is another important metric. If air quality levels are measured every week, or every hour of the day, makes a difference at the time of taking decisions. This can also be added to the utility function in question.

6.3 Utility Metrics for the OpenIoT Digital Agriculture Phenonet Use Case

The Digital Agriculture – *Phenonet* – implemented by CSIRO in the course of WP6 could make use of some of the utility metrics described in previous chapters. 'Phenonet' describes the network of wireless sensor nodes collecting information over a field of experimental crops. The term “Phenomics” describes the study of how the genetic makeup of an organism determines its appearance, function, growth and performance. Plant phenomics is a cross-disciplinary approach, studying the connection from cell to leaf to whole plant and from crop to canopy (CSIROa 2013).

Analysing the size, growth and performance of plants in a greenhouse or field site can be time-consuming and laborious. More specifically, when a field site is located in a remote area, it becomes quite expensive to send people out to the field. The ability to collect this information from remote locations and send it back to the laboratory in real time is an invaluable tool for plant scientists (CSIROa 2013, CSIROb 2013).

CSIRO has developed smart wireless sensors nodes that work autonomously and independently cooperating with each other to set up an ad hoc network to record environmental conditions and wirelessly transfer data to a data store.

The Phenonet project is supported by a production system with commercial quality grade software and unit tests developed for researchers in CSIRO and government organizations in Australia. Phenonet platform is currently being used on a daily basis and enjoys high level of uptime and very stable code. The high-level architecture of Phenonet project is depicted in Figure 10 and consists of five stages.

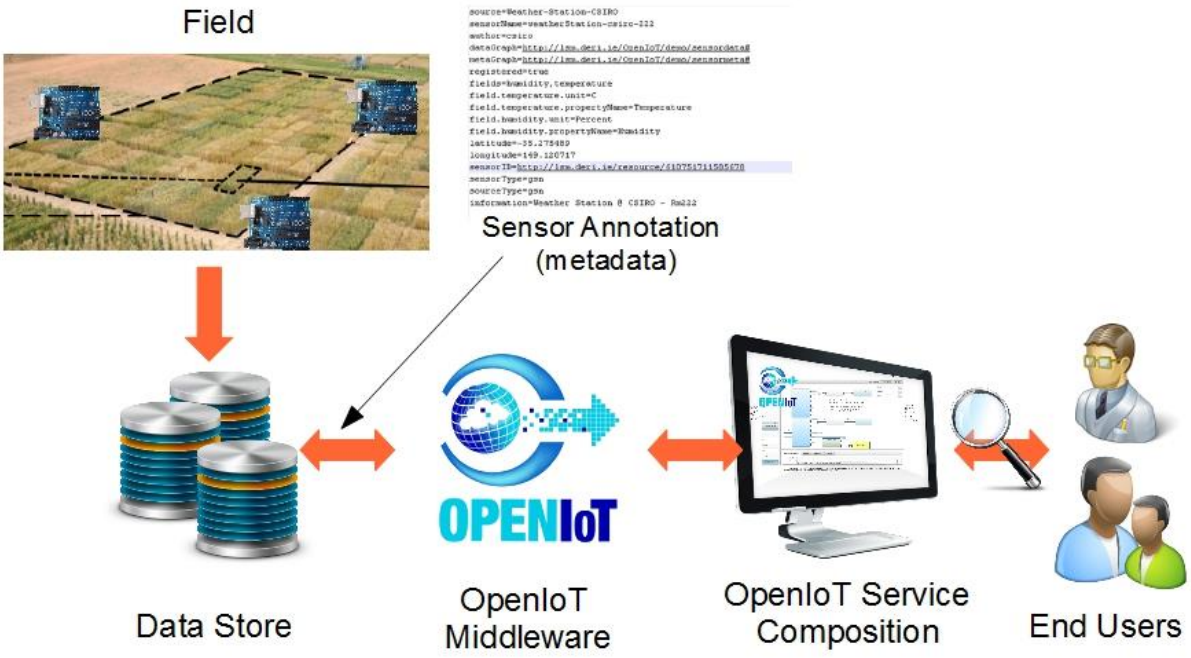

Figure 10: Phenonet Architecture based on OpenIoT platform

- **Field**: The field is an experimental plot comprising different types of crops varieties. Wireless sensors are installed in the experimental plots that measure various environmental features such as soil temperature, crop canopy temperature, humidity, wind speed etc. Using this information, the crops growth, performance, size, etc. are continuous sensed/computed in real-time.
- **Data Store**: Data Storage highlights the need to have all captured data and information about the data (metadata), to be stored in a safe location. At storage state, we are targeting both sensor measurements and metadata information. Examples of metadata information include; sensor types, serial numbers, MAC address, experimental treatment, crop sowing date, genotype, replicate number etc. Each sensor stream is identified using a globally unique identifier (GUID). This layer in current Phenonet relies on python scripts to upload data into the system.
- **OpenIoT middleware to enable Data Analysis**: Is the brain behind all the calculations, data modelling, data cleansing and linear aggregation models used by Phenonet project. This component directly contacts Data Store layer when it requires data from a particular stream. Internally, Data Analysis component also performs extensive caching and applies proprietary algorithms and mechanisms to ensure a highly responsive interaction with the system is maintained at all times. Data Analysis component is accessible through HTTP RESTful API. The response to any request received by this component is in format of JSON object. This layer is developed in Scala.
- **OpenIoT Service Composition using HTML5 Visualization**: This component is responsible to generate RESTful network requests and send them to data analysis component. The response is then rendered by the frontend and appropriate visualization components. This layer is written in CoffeeScript and uses HTML5 for rendering and visualization.

- **End user**: Ranges from a plant biologist to a farmer. The system also provides tools and mechanisms to share data analysis and visualizations with other group of users.

For such a complex system as Phenonet potentially a large number of utility metrics can be used. However, we identify and focus on the following important metrics:

- **Data quality**: Sensors deployed in the field are continuously producing data streams. GRDC, Australia plants 1 million $10m^2$ plots every year, and each plot produces about 50,000 data points every week. Given the economy of scale we are potentially looking at big data. However, only a fraction of that data constitutes what is called “golden data points”. Therefore, it is crucially important to define data quality utility metric and have a reliable way of computing it.
- **Data Capture (Cost of data collection and maintenance):** The Phenonet infrastructure consists of a large number of sensors and weather stations deployed in the field. Some of the sensors are wireless and require regular replacement of batteries. All of the sensors also require regular calibration. These all carry some costs that need to be computed and considered when taking into account the data capture cost.
- **Cost of data analytics models:** The Phenonet data needs to be processed, validated, analysed. Different types of processing require different degrees of complexity which can be described by the utility metrics.
- **Return on investment (RoI):** In order to encourage farmers to invest in the Phenonet infrastructure and share the data for the common good as well as a viable and attractive incentive, the RoI metric should be a convincing motivation for investing into the Phenonet infrastructure.

7 IOT BILLING BASED ON OPENIOT UTILITY METRICS

7.1 Overview

In this section we describe a simple pay-as-you-go framework that can be integrated (as an add-on module) to the OpenIoT platform. A main characteristic of the framework is that it uses utility metrics and functions of the OpenIoT ontology (described in deliverable D3.1) in order to implement the ever important billing functionality. Though simple, this framework is among the very few efforts towards implementing practical billing schemes for IoT. The framework includes three different sets of data and related data structures:

(a) User information data
(b) OpenIoT usage data, which are derived from the OpenIoT platform (and its ontology) via a REST Web Service and
(c) Customer invoice and billing data, which are essential elements of the billing functionality.

In the sequel we present the main data structures used to store and process the corresponding data as long as the data flow between the Service Delivery Management module and the Utility Management module.

7.2 Data Structures

7.2.1 User information data

Figure 11 shows the data structures used to store the user information data, and the corresponding relations between them. Each new user in the system triggers the generation of a new BaseUser object class. Each object instance is tied to a single row in a database table. The creation of a new object corresponds to the addition of a new row upon save, or the update of an existing row upon update. The class fields are as follows:

- id : The unique user identifier. It corresponds to the database table primary key.
- createDatetime : The creation time of the user. That corresponds to the time the user was inserted to the database.
- failedAttemps : The number of failed login attempts. After a number of failed attempts that are defined by the system administrator, the user is locked and should be unlocked by the administrator.
- lastLogin : The time the last login of the user occurred.
- lastStatusChange : The time the last status change of the user occurred.
- password : The user password.
- userName : The user name. That corresponds to a valid email.
- firstName : The user first name.
- lastName : the user last name.
- baseEntity : BaseEntity class object. It is used to identify the user as a customer or an administrator.\

- currency : Currency class object. The implementation supports a multi-currency system. It is used to identify the user choice of currency.
- status : GenericStatus class object. It is used to identify the user's activity status. The possible values are: (a) Active user; (b) Inactive user; (c) Locked user.
- language: Language class object. The implementation supports a multi-language system. It is used to identify the user choice of language.

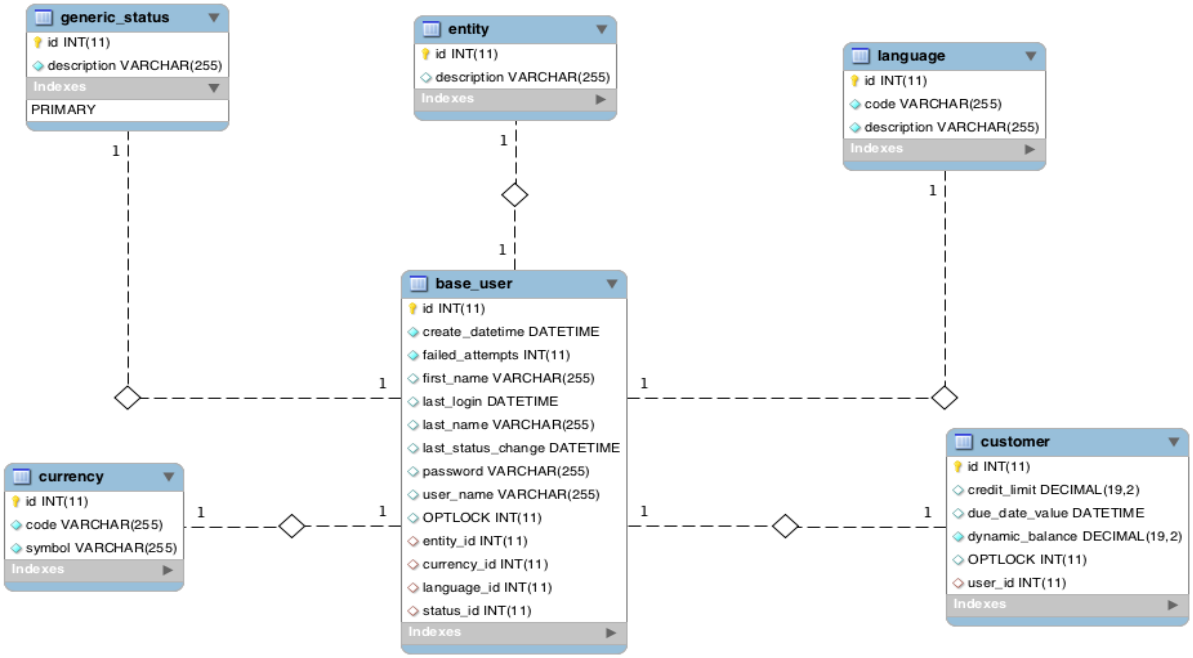

Figure 11: User Information Data Structures (part of the OpenIoT Add-on Billing Framework)

For customers of IoT services, we use additional fields to store customer only related data. The class fields are as follows:

- id: the user last name. The unique user identifier. It corresponds to the database table primary key.
- creditLimit : The customer credit limit, that is set by the system administrator.
- dueDateValue : The due date of the customer's current payment instalment.
- dynamicBalance : The customer's current balance. It is updated each time the customer orders a new service or makes a new payment.
- baseUser : BaseUser class object. That is used to link the customer object with the corresponding BaseUser object.

7.2.2 OpenIoT usage data

The OpenIoT usage data are handled with the use of three related classes. Those are the "OpenIoTApplication" class, the "OpenIoTService" class and the "OpenIoTSensor" class. The three classes are presented in Figure 12. Each user

request is server by an OpenIoT application. In order to fulfil the user request, multiple OpenIoT services may be needed, therefore an OpenIoT application may comprise of multiple OpenIoT services. Finally, OpenIoT service may gather data from multiple OpenIoT sensors. Following, we will describe the three classe's fields.

The "OpenIoTApplication" class fields are as follows:

- applicationId : The application Id that is the primary key in the database table.
- requestDatetime : The application request time by the customer.
- applicationCost : The total cost of the application. This is calculated after the choice of the billing scheme is applied.
- status : It describes the order status to which the IoT application corresponds to. The possible values are: (a) New order; (b) Order invoice created; (c) Order completed; and (d) Order cancelled.
- baseUser : BaseUser class object. This is the object of the user that requested the IoTApplication.
- billingScheme : BaseEntity class object. This is the object that has the description of the billing scheme. The billing scheme will be applied during the application cost calculation.

The "OpenIoTService" class fields are as follows:

- serviceDesc : The service description that is the primary key in the database table.
- dataVolume : The total volume of data used by the service, in order to accommodate the application's request.
- dataType : The type of data measured in the dataVolume field. That can be time usage or volume usage.

The OpenIoTSensor class fields are as follows:

- sensorId and serviceDesc : The sensor Id and the service description, comprise of a composite primary key. They identify the sensor and service the data correspond to.
- sensorWeight : The sensor weight that identifies the importance of the data provided by the sensor. That is taken into account during billing.
- BillingSchemeCost: The Cost per billing Scheme

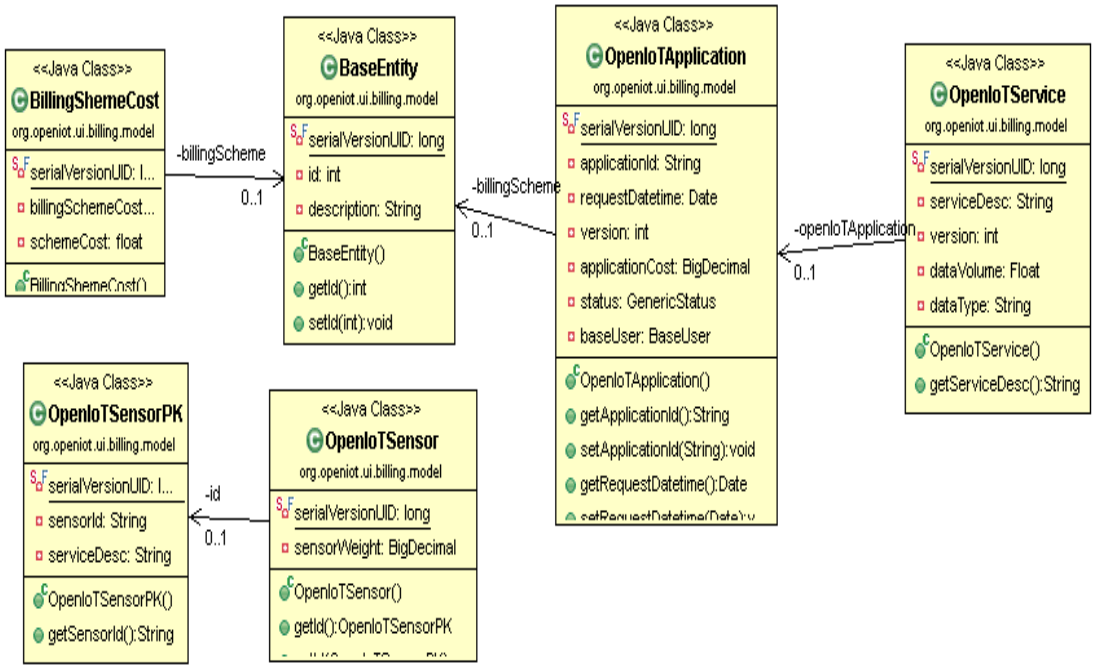

Figure 12: OpenIoT Usage/Utility Data

7.2.3 Customer invoice and billing data

Customer invoice follows the information recorded at “OpenIoTApplication”. Each “OpenIoTApplication” request corresponds to a single invoice line. All invoice lines corresponding to a single customer request will form an invoice. Each customer will have multiple invoice lines, each one corresponding to a single request. Each invoice line information is stored in an “InvoiceLine” class object, while each invoice information is stored in an Invoice class object. Both classes are presented in Figure 13.

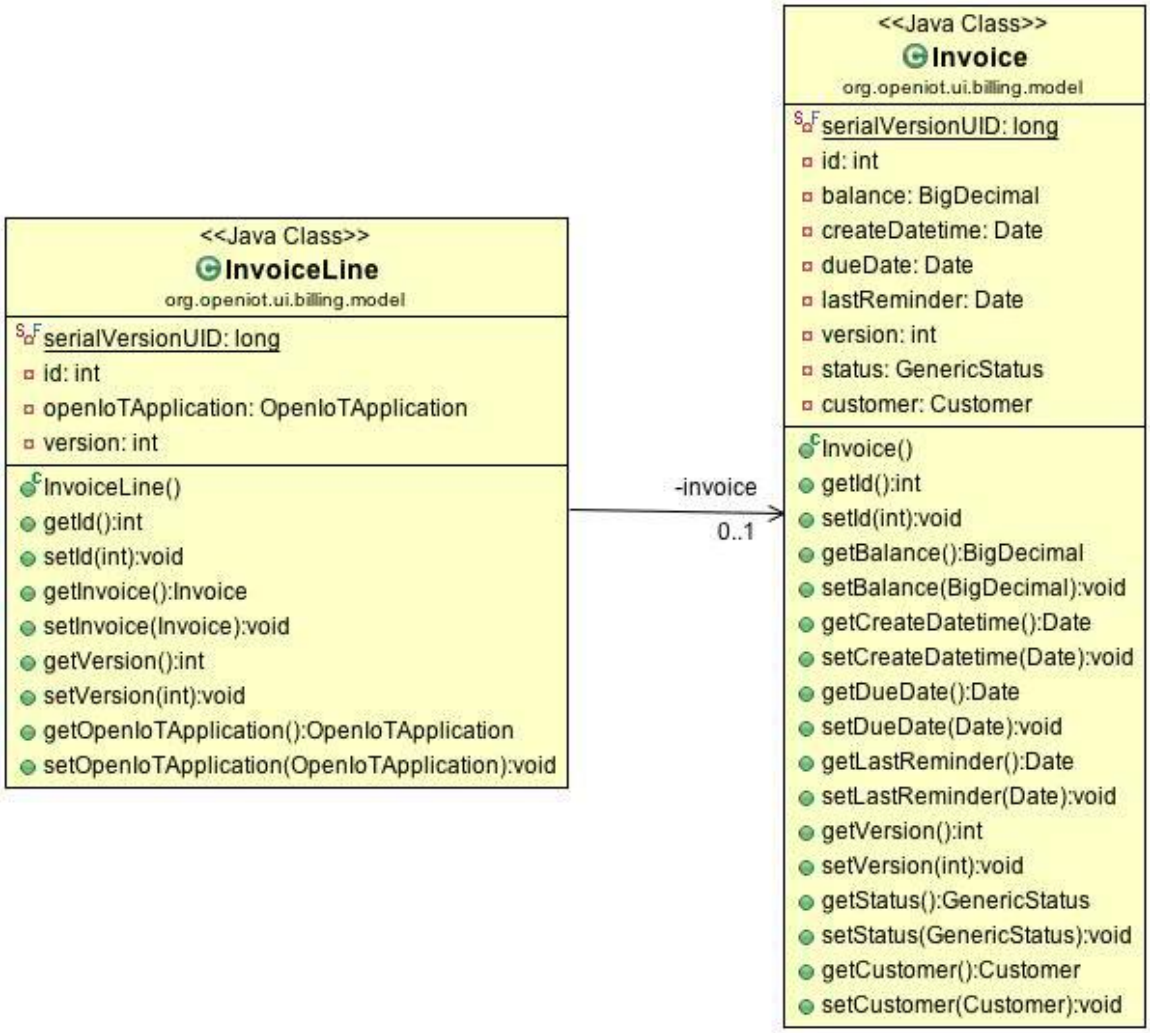

Figure 13: Invoice and Billing Data of the OpenIoT Billing Framework

The InvoiceLine class fields are as follows:

- id : The invoice line id that is the primary key in the database table.
- openIoTApplication : The OpenIoTApplication usage corresponding to the invoice line.
- invoice : Invoice class object. The invoice to which the invoice line corresponds to.
- The Invoice class fields are as follows:
- id : The invoice id that is the primary key in the database table.
- Balance : The invoice balance. It corresponds to the total cost of the invoice. When a payment of the invoice is performed the balance is decreased by the amount paid.
- createDatetime : The date and time the invoice was created.
- dueDate : The date and time until which the customer should pay the invoice.
- lastReminder : The date and time the last reminder for the payment was send to the customer.
- status : The status of the invoice. Possible values are: (a) Invoice created; (b)Invoice cancelled; (c) Payment successful; (d) Payment failed; and (e)Party paid.

- customer : Customer class object. The customer to which the invoice corresponds to.

In order to process payments, each invoice corresponds to a single payment invoice. The customer will be presented with all the pending payment invoices, and he will choose with which ones to proceed. The chosen ones will be handled as a single payment comprising of the chosen payment lines. Each payment invoice information is stored in a PaymentInvoice class object, while each payment information is stored in a Payment class object. Both classes are presented in Figure 14.

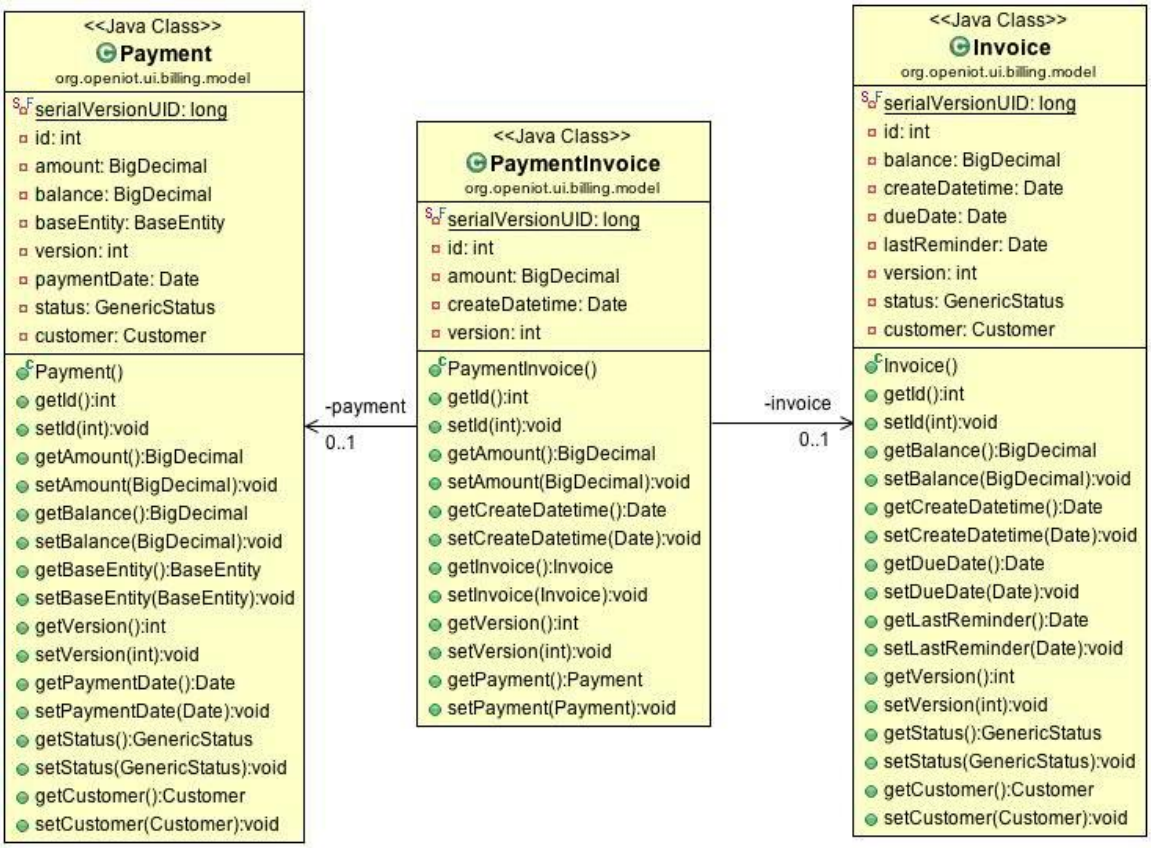

Figure 14 Customer payment data structures

The PaymentInvoice class fields are as follows:

- id : The payment invoice id that is the primary key in the database table.
- amount : The amount corresponding to the payment invoice.
- createDatetime : The date and time the payment invoice was created.
- invoice : Invoice class object. The invoice to which the payment invoice corresponds to.
- payment : Payment class object. The payment which the payment invoice is part of.

The Payment class fields are as follows:

- id : The payment invoice id that is the primary key in the database table.
- amount : The total amount corresponding to the payment.
- balance : The payment balance. That is used in case the customer does not pay the total amount; therefore balance represents the outstanding amount.
- baseEntity : BaseEntity class object. It stores the payment method. The available methods are: (a) Payment by credit card; (b) Payment by bank transfer; and (c) Payment by Paypal.
- paymentDate : The date and time of the payment.
- status : GenericStatus class object. The possible values are: (a) Payment successful; (b) Payment failed; and (c) Party paid.
- customer : Customer class object. The customer related to the payment.

The customer invoice and billing data schema is shown in Figure 15.

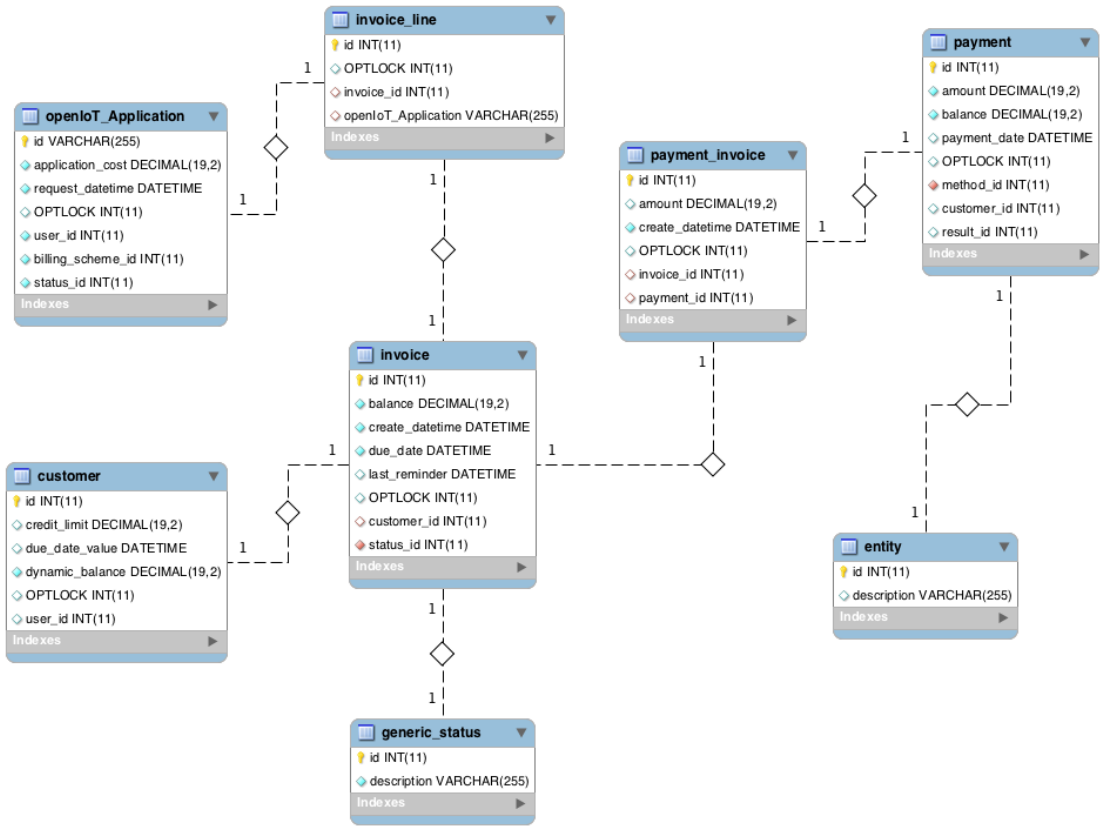

Figure 15 Customer invoice and billing data schema

7.3 Utility Metering Process

Metering is performed in the Service Delivery & Utility Management module. The module enables data retrieval from the selected sensors comprising the OpenIoT service, and maintains and retrieves information structures regarding service usage. When a service is requested, the module performs the following steps:

- Delivers the requested service

- Access and process data streams from the cloud, following processing instructions specified during the request
- Keep track of utility parameters associated with the request, like the time usage of the service, the transmitted data volume and number and type of sensors used.
- Encapsulate this info to the “AppUsageReport” document (shown in Figure 16) and forward it to the Utility Management module.

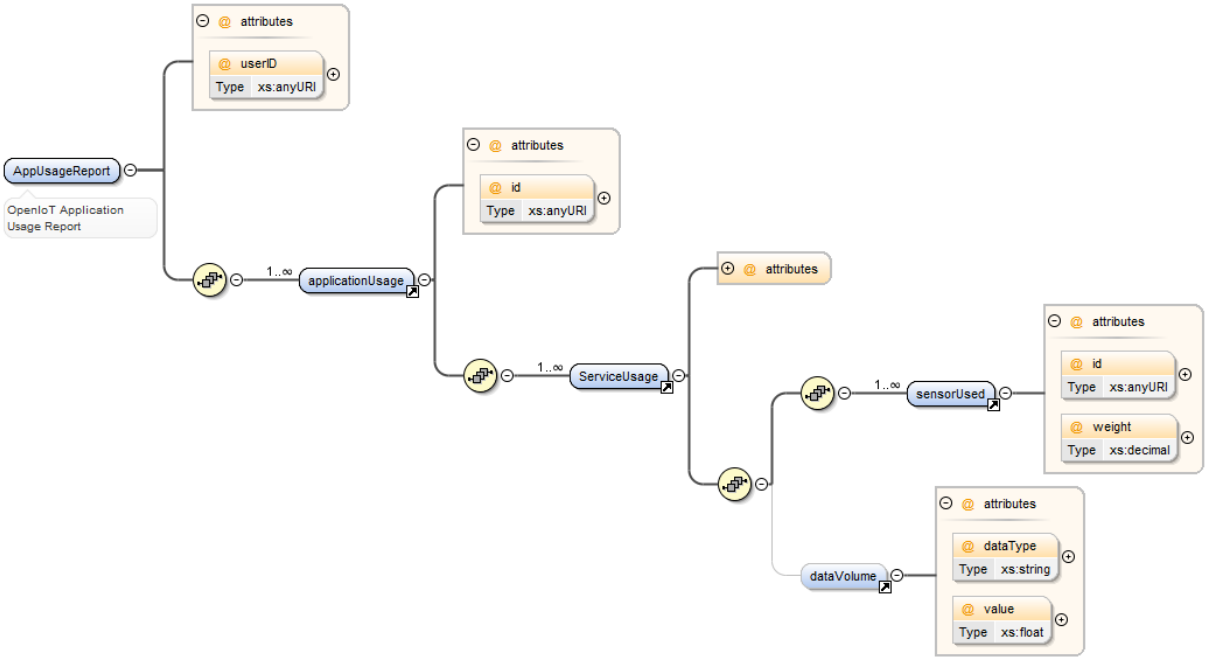

Figure 16 App Usage Report Object

The Utility Management system reads all the info provided by the pay-as-you go framework in the Database. An administrator is needed to assign billing scheme to an OpenIoT User and after that the utility calculates according to the billing scheme the price per invoice-line and bills the user. The user can login and download in pdf his invoice and see per “OpenIotApplication” the sensors the he have used and billed for.

7.4 Scenarios Validation

In this section we present the application of three billing schemes in the pay-as-you-go framework. The billing schemes examined is the following:

a) flat-rate scheme
b) time-based scheme and
c) volume-based scheme.

The application of the flat-rate scheme is the simplest of all, since it is calculated on the basis of a fixed tariff for a specified amount of time (i.e per month, per annum, etc.). In order to calculate billing with the use of that scheme, we do not really need

any utility metrics, since the customer will be changed with the flat-rate amount. The only validation we would need to perform would be to verify that the customer paid the corresponding instalment for the service.

In order to apply the time-based or volume-based scheme, we use the recorded dataVolume per service, as shown in Figure 16 & Figure 17. In case we use a time-based scheme the measurement base unit will be time (i.e seconds), while if we use a volume-based scheme the measurement base unit will be the volume of date (i.e Megabytes). In order to calculate the total cost of the customer request, we gather all the application requests. For each application, we add the time or data-volume used per service, and calculate the total time or data volume used by the application. The customer is billed based on the time or data volume used by all the applications he requested, in combination with the service change per time unit or data-volume unit. Additionally, we can also apply the weight of each sensor in the service usage, which is also recorded in the system. That will provide more accurate usage calculation per sensor per service, and accordingly more accurate billing.

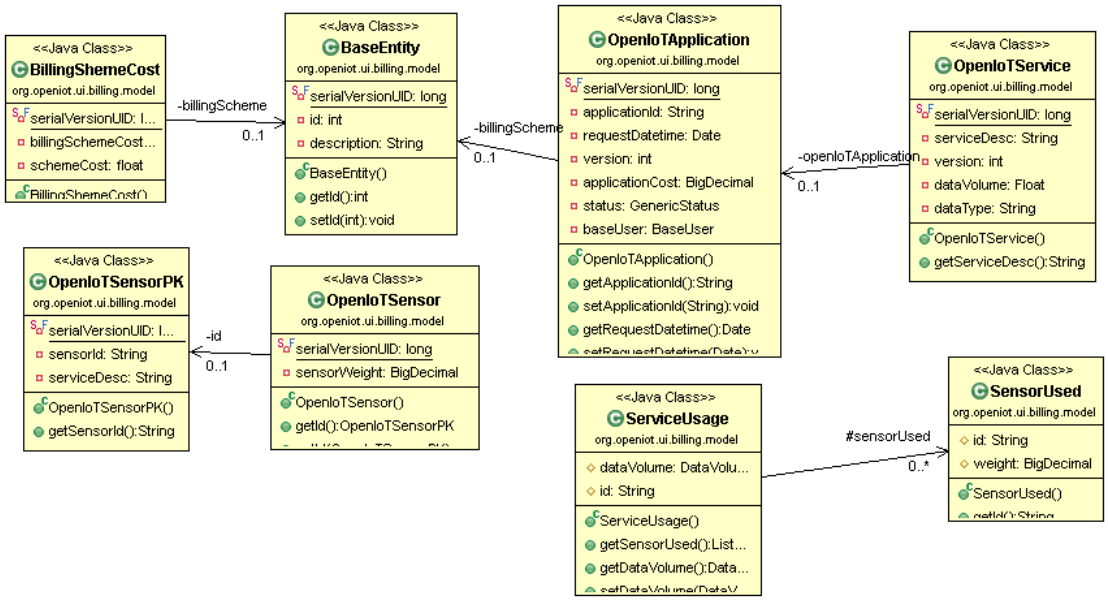

Figure 17 Billing Scheme Assignment class structure

8 CONCLUSIONS

This deliverable has specified a set of utility metrics that will be recorded and processed in the scope of the OpenIoT platform implementation. These involve metrics pertaining to physical devices (e.g., the quality of a sensor, the energy consumption of a wireless sensor, the bandwidth of the sensors, the data volume produced and/or consumed by a given sensor, the type of the sensor, the location of the sensor), as well as metrics pertaining to virtual sensors (which lack properties that pertain to the physical characteristics of the sensor). Furthermore, the deliverable has outlined several schemes (e.g., volume based, time based, SLA based) that take advantage of metrics for individual sensors in order to build metrics meaningful for wider IoT applications and services. In this context some concrete examples of utility calculation associated with the manufacturing use case of the project have been given.

Overall the **utility metrics** identified in this deliverable can give rise to the implementation of several **metering, accounting and billing schemes**. These may include schemes that have already been proposed in literature (along with their variations), but possible also additional schemes that could be introduced in response to specific application requirements. Moreover, the utility metrics that are specified in this document provide the basis **to the implementation of a range of utilitarian optimization techniques (WP5)**, which attempt to optimize specific metrics during the course of the operation of multiple IoT services within the OpenIoT platform. As a result, the OpenIoT platforms should not be confined to keeping track of a subset of metrics that are relevant to the proof-of-concept implementation of the OpenIoT platform and use cases. Rather, it should record and keep track of the full range of sensor utility metrics that are specified in this document. By recording these metrics, the OpenIoT platform will allow integrators and services providers to implement metering and billing schemes of their choice. These schemes will be applicable to the service and applications levels.

In this document, we have also detailed how these metrics can **be used to formulate utility functions** that encompass different query requirements from users and applications, while constraint to a certain budget. Beyond the state of the art in this area, we have shown how this can be done even for very heterogeneous requests form the users, and we have shown specific optimization techniques, targeting the selection of only the most relevant sensors from the complete set of available ICOs.

Note that the recording of the various utility metrics are in the process of being implemented at the Utility Manager module of the OpenIoT platform. This module stores, updates and manages the various metrics associated with the various sensors, while providing the means for combining them towards service-level utility calculation. Specifically, we have described how a utility-based optimization approach has been designed for the OpenIoT architecture, such that it is able to efficiently use subsets of the available virtual sensors exposed by X-GSN. As a result, X-GSN at the local scheduling level can actively activate and deactivate sensors, if they contribute or not to the overall social welfare. This optimization scheme has been detailed in Deliverable D5.1.2 in detail. Historical information associated with the various metrics should be also maintained, since this can be required by some schemes, while also being useful for monitoring purposes.

Finally, we have described a **simple pay-as-you-go framework** that can be integrated to the OpenIoT platform. A main characteristic of the framework is that it **uses utility metrics and functions** of the OpenIoT ontology.

5 REFERENCES

[Choi 2007] Choi, M.-J. and Hong, J. W.-K. "Towards Management of Next Generation Networks," IEICE Transaction Communications (E90- B:11), 2007, pp. 3004-3014.

[CSIROa] CSIRO, 2013 Phenonet: wireless sensors in agriculture. Available from: http://www.csiro.au/Outcomes/ICT-and-Services/National-Challenges/Wireless-sensors-in-agriculture.aspx. Accessed on Dec 2013

[CSIROb] CSIRO, 2013 The High Resolution Plant Phenomics Centre. Available from: http://www.csiro.au/Outcomes/Food-and-Agriculture/HRPPC/Sensors-in-the-field.aspx. Accessed on Dec 2013

[Dunkels 2007] A. Dunkels, F. Osterlind, N. Tsiftes, and Z. He, “Software-based on-line energy estimation for sensor nodes,” in Proceedings of the 4th workshop on Embedded networked sensors(EMNETS’07). New York, NY, USA: ACM, 2007, pp. 28–32.

[Falkner 2000] M. Falkner, M. Devetsikiotis, I. Lambadaris: An Overview of Pricing Concepts for Broadband IP Networks. IEEE Communications Surveys, Second Quarter (2000)

[Krause 2008] A. Krause, E. Horvitz, A. Kansal, and F. Zhao, “Toward community sensing,” in Proceedings of the 7th international conference on Information processing in sensor networks , ser. IPSN ’08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 481–492.

[Liu 2008] H. Liu, B. Krishnamachari, and M. Annavaram, “Game theoretic approach to location sharing with privacy in a community-based mobile safety application,” in Proceedings of the 11th international symposium on Modeling, analysis and simulation of wireless and mobile systems, ser. MSWiM ’08. New York, NY, USA: ACM, 2008, pp. 229–238.

[Odlyzko 1999] Andrew Odlyzko, «Paris-Metro Charging for the Internet», EC '99 Proceedings of the 1st ACM conference on Electronic commerce, Pages 140-147

[Riahi 2013] M. Riahi, T. G. Papaioannou, I. Trummer and K. Aberer. Utility-driven Data Acquisition in Participatory Sensing. 16th International Conference on Extending Database Technology (EDBT), Genoa, Italy, March 18-22, 2013.

[Schmidt 2007] D. Schmidt, M. Kr¨amer, T. Kuhn, and N. Wehn, “Energy modelling in sensor networks,” Advances in Radio Science, vol. 5, pp. 347–351, Jun. 2007.

[Shnayder 2004] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh, “Simulating the power consumption of large-scale sensor network applications,” in SenSys ’04: Proceedings of the 2nd international conference on Embedded networked sensor systems. New York, NY, USA: ACM Press, Nov. 2004, pp. 188–200.

[Tse 1998] David N. C. Tse, Stephen V. Hanly: Effective Bandwidths in Wireless Networks with Multiuser Receivers. INFOCOM 1998: 35-42

[Xie 2009] X. Xie, H. Chen, and H. Wu, “Bargain-based stimulation mechanism for selfish mobile nodes in participatory sensing network,” in Proceedings of the 6th Annual IEEE communications society conference on Sensor, Mesh and Ad Hoc Communications and Networks , ser. SECON’09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 72–80.

