Final Publishable Summary Report

Grant Agreement number: ***317674***

Project acronym: ***BETAAS***

Project title: ***Building the Environment for the Things as a Service***

Funding Scheme: ***Collaborative Project***

Period covered: **from** **01/10/2012** **to** **31/03/2015**

Name of the scientific representative of the project's co-ordinator, Title and Organisation:

Ms. Novella Buonaccorsi

Program Manager for

INTECS S.p.A

Via U. Forti, 5A

56126 PISA

Tel: ***+39.050.96.57.411***

Fax: ***+39.050.96.57.400***

E-mail:
novella.buonaccorsi@intecs.it

Project website address: **http://www.betaas.eu/**

1. Executive summary

This document summarizes the key aspects, the context and the achievements of the BETaaS project.

BETaaS: Building the Environment for the Things-as-a-Service is an European project founded under the 7th Framework Programme. The BETaaS platform provides a unified framework for the development of Machine-to-Machine (M2M) applications through a content-centric service-oriented interface, named Thing-as-a-Service. BETaaS also **exploits the benefit provided by the use of Semantic technologies in the IoT context and enables the emergence of high level interaction and collaboration of things and platforms through the BETaaS TaaS Reference Model**. This is expected to have a great impact in providing a high level integration among different platforms (mobile, devices and services) making very attractive the opportunity to build open source models over a combination of heterogeneous systems.

Another relevant aspect in which **BETaaS also provides an added value, concerns the future of hardware hubs for the IoT**. The need of gateways or hubs that connect multiple things and enable a closer layer of service is ramping up. Actually there are several vendors providing specific vertical appliances for different scenarios, especially in the smart home domain (automation, energy and security). These specialized gateways are going to increase the number of contexts in use and they will even embrace new kind of Things in the future. In this scenario, developers and vendors are demanding platforms that abstract from the myriad of heterogeneous systems and different underlying technologies. For this purpose, BETaaS is able to run simultaneously on multiple heterogeneous hardware platforms, easily adapting to an unlimited set of protocols for Things communication and allowing to accept context information belonging to a lot of different domains.

A key feature of BETaaS is about putting the "cloud intelligence" close to the data source and to the interaction among Things. This not only decreases data latency, but also reduces network unavailability issues, speeds-up smart autonomous-responses and Big Data capabilities, using the concept of what is called "Fog Computing". It may also be possible to provide better security, privacy and control over sensitive devices and data within a localized environment.

BETaaS is built to take the opportunity to provide an answer to some of the key trends and drivers found during the project and ruling the economy and the expected influence of IoT. The expected evolution in the today's world is that the same global mobile and internet services leaders that now dominate the consumer battle are the best positioned to provide the platform-centric worlds that the IoT will likely need. Therefore the emergence of platforms and open standards (intentional or unintentional) will come into place, which will accelerate the pace of innovation.

2. Summary description of project context and objectives

2.1. Project context

Today there are countless devices at work to improve productivity and quality of life of human beings, in all technological domains. In most cases they operate in isolation or with very little cooperation from their likes, and serve a well-defined single purpose for which they have been engineered. Cheaper sensors and more powerful devices, reduced communication and network costs with higher bandwidth and ubiquitous coverage options (e.g. 4G and WiFi networks), smartphones evolution and penetration and the adoption of IPv6 to assure the creation of a network made of millions of devices, this will provide an unprecedented opportunity to create applications and services that go far beyond the mere purpose of each participant, as enabled by the interactions in this so-called Internet of Things (IoT). In principle, all the things of the world can be empowered to become "smart" and communicate with other peers or remote systems to better serve their purpose. A common aspect of IoT applications is that many of them of practical interest involve control and monitoring functions, where human-in-the-loop actions are not required. As a matter of fact, the only reason for having many of these applications is to remove human intervention for improved efficiency, security, and safety.

There is a strong need of expanding network availability and connectivity in order to enable the adoption of devices to the network. In a very close future there will be a myriad of devices producing tons of data that could be used to make our lives easier and better, to improve efficiency, to improve business and ordinary life processes, or even to create new ones to fully leveraging the potential of the IoT. This scenario is also likely to produce a positive feedback and therefore, in a few years from now, we're expecting to have millions of smart connected Things and systems into our networks. What we have to do is to make them become part of the choreography. They will be part of a coordinated part of a giant programmable machine put to the services for humans and business. Software will be a big part of the IoT market. This would be like a Programmable World in which applications will thrive to reach to the full potential of the IoT.

The barriers to enter the M2M segment today are high, since the market is very fragmented and user requirements and expectations are very heterogeneous. This scenario is especially harsh for small and medium players, who do not have a sufficiently vast commercial influence or the strength for massive marketing campaigns. By defining open interfaces, which will be also pushed for standardization in the relevant technical committees, M2M service providers have reduced risks of the investment and less training/setup costs, since they can re-use the skills and experience over different projects and also across various domains.

2.2. Objectives

BETaaS focuses specifically on the Machine-to-Machine (M2M) applications which will create a bridge between the real world (made of sensors, actuators, tags that are pervasive in our lives) and the virtual world (the Internet and its associated services).

BETaaS is creating the fertile soil for this change through the definition of a horizontal runtime platform to be released as open source software and designed for the future IoT. In addition, it is designed to allow both easy integration of existing vertical systems and the development of custom services (*extended services*), and provides built-in support for non-functional requirements such as QoS, big data management, and security. Finally, it relies upon a distributed runtime environment made of a local cloud of nodes that allows accessing smart objects connected to the platform regardless of their technology and physical location.

To overcome the limitations of the current systems for M2M applications, a novel approach is proposed based on the following principles: 1. Storage and processing of data are as close as possible, in space and time, to where they are generated and consumed. 2. Important non-functional requirements, namely security, reliability and QoS, are supported by means of a tight integration of the high-level services provided with the physical resources of the peripheral devices, i.e., things and gateways. 3. Energy efficiency and scalability of the systems are achieved through the distribution of on-the-spot inferred content, rather than raw data. These principles are designed and implemented through a content-centric platform distributed over a local cloud, hosted by the gateways and providing an environment for applications accessing M2M services and devices through a set of services. Its deployment is dynamic so as to follow the time-varying M2M services and the changing characteristics of the applications.

The proposed platform provides uniform interfaces and services to map content (information) with Things (resources) in a context-aware fashion. Deployment of services for the execution of applications is dynamic and takes into account the computational resources of the low-end physical devices used. To this aim, the platform is based on a suitable defined IoT model, which allows the integration of the BETaaS components within the future Internet environment.

The BETaaS platform is released as open source to achieve the different goals: i) to benefit from the contributions of the open source community of developers (in terms of customization, testing, developments, …); ii) to allow M2M service providers to focus on the application-specific aspects, without the need for working on common features, which reduces the development costs and time-to-market. Actually, the nature of the platform and its architecture play a key role in encouraging several actors in contributing to its development. Actors range from private users willing to exploit their own devices to make up complete customized systems, to service providers interested in developing specific solutions through the creation of extended services, to public municipalities aiming at publishing basic and derived information coming from Things deployed on the territory.

The platform then aims at reaching several classes of intermediate and final users and it has been designed to make this process easier. There are different points of the platform architecture that go in that direction. For example, its capability of adaptation to new Things technologies encourages devices vendors to provide simple software packages in order to allow the dissemination of their solutions. The possibility to create customized extended services is suitable to prepare ready-to-use packages based on BETaaS and enriched with specific logic to adapt to specific domains or customers. High level interfaces with external applications, mainly based on context information rather than on detailed references to hardware devices, make easier the development of apps. That is even more true if we consider the standard mechanisms that are implemented to provide such interfaces.

In order to provide to platform developers a mean to easily perform the first experiments with it, a Thing Simulator has been developed and included in the open-source release. In this way developers may start experimenting with the platform or developing their own apps without the need of buying real devices. Furthermore the Simulator can even be used in conjunction to real devices and that could be very useful also to is interested in testing large number of Things before putting the platform in the operational environment. As an objective of the platform is to reach as many developers as possible giving them the possibility to instantly make experiments. A CoAP adaptor plug-in is also included in the open-source release. That module will surely allow a lot of people to start using their own CoAP-enabled devices.

3. Main Scientific & Technical results/foregrounds

3.1. Implementation: a local cloud distributed platform

The BETaaS platform implementation is based on the *OSGi*[1] *technology*: BETaaS services are designed to be modular and highly dynamic, and OSGi is a framework that satisfies such requirements; in fact it allows the deployment of bundles that expose services discoverable and accessible through a service registry, provided by the OSGi container itself. The OSGi service registry is restricted to a local container without possibility of sharing services: in order to overcome such limitation, BETaaS takes also advantage of Distributed OSGi, which allows sharing OSGi services between different containers through a distributed registry, implemented by using Apache Zookeeper and Web services. The BETaaS platform, in order to offer a seamless deployment mechanism, also leverages Apache Karaf as OSGi container. Karaf allows the provisioning of bundles groups by using features. A feature is actually a list of bundles with their related dependencies, which can be then deployed inside the container directly from the BETaaS repository. In this way, the deployment of a BETaaS gateway can be performed through a feature or a set of features, without requesting any manual installation of a bundle and its dependencies.
Platform capabilities are implemented as bundles in order to exploit the modularity provided by OSGi.
Context Awareness Look-up is implemented by the *Context Manager* (CM) bundle. Every gateway has its own CM, which contains a BETaaS ontology and a Semantic Parser. Through these two elements, the CM *is able to unify the information coming from heterogeneous resources and applications, and to infer knowledge from raw data in a context-aware fashion*.
Whenever a thing is connected to a gateway, contextual information about this thing (location, type, etc.) is automatically or manually retrieved by the Adaptation Layer of the gateway. This information is sent to the Semantic Parser of the gateway. In order to promote standardization, the Semantic Parser uses WordNet[2] to translate this contextual information to WordNet synsets whenever possible, and stores this data in the ontology.
Using the semantic requirements of the Manifest file, the *CM creates a thing service for each of the things connected to a gateway*. The information of the things registered in a gateway is propagated in the instance by means of the TaaSRM (the bundle that implements the TaaS layer in every gateway), which communicates locally with its own CM.
The *QoS Manger* bundle implements QoS functionalities. At the TaaS layer the module adopts the WS-Agreement Negotiation protocol, which has been implemented leveraging on an existing publicly available implementation, WSAG4J a java-based implementation. The implementation has been customized in order to store information within the distributed data storage embedded in the platform.
Security management capabilities are implemented within the *Security Manager* bundle. The implementation of digital certificate has been performed based on Java cryptography library, namely *bouncycastle*[3].
Big Data Manager bundle provides services to store data in a SQL database, e.g. MariaDB, H2 and Mysql. With respect to the analytics platform, it uses the Apache Sqoop2 server to load data from a SQL database into a Hadoop HDFS. Big Data Manager also uses Apache Hive Metastore to define a metatable on top of the HDFS imported data: leveraging Hive, such data can be then processed by a data task, through the usage of the PrestoDB query system.
Finally, *Virtual Manager* bundle provides virtualization capability. Its implementation relies on the usage of livbirt as the way to manage local VMs, both for x86/x64 and ARM architectures, thanks to

[1] http://www.osgi.org
[2] http://wordnetweb.princeton.edu
[3] https://www.bouncycastle.org/java.html

the last versions of Xen hypervisor. Moreover, support for clouds built on OpenStack (through its API libraries) and OpenNebula (through OCCI) is included.

3.2. TaaS reference model

The BETaaS platform aims at providing a runtime environment relying on a local cloud of gateways to support the deployment and execution of content-centric M2M applications. The platform seamlessly integrates existing heterogeneous M2M systems, which constitute the so-called underlying physical layer, consisting of smart things connected to a global network infrastructure and providing basic services. In order to account for such heterogeneity, BETaaS builds upon a baseline reference model and architecture, which we call Things-as-a-Service (TaaS), providing an abstract and uniform description of the underlying physical layer [1] [2] .

Following a standard approach, the BETaaS reference model is comprised of five models: domain, information, communication, security and functional, respectively. The IoT-A reference model provides a basis for all such models. In particular, BETaaS adopts the domain model of the IoT-A project, and models the data managed by the platform by means of ontologies aligned with the IoT-A information model. More specifically, in order to unify the information that comes from heterogeneous resources and applications, and to infer knowledge from this raw data, a BETaaS ontology is defined based on several existing ontologies from the IoT domain that fit our requirements. The IoT-A domain model is then used as an Upper Ontology of the network of ontologies, i.e., the BETaaS ontology, which is implemented to model the data [3] .

BETaaS exploits the IoT-A communication and security models almost straightforwardly, while the functional model was revised and extended in order to reflect the specific requirements of the BETaaS platform. In particular, the model follows a layered approach comprising four layers: the

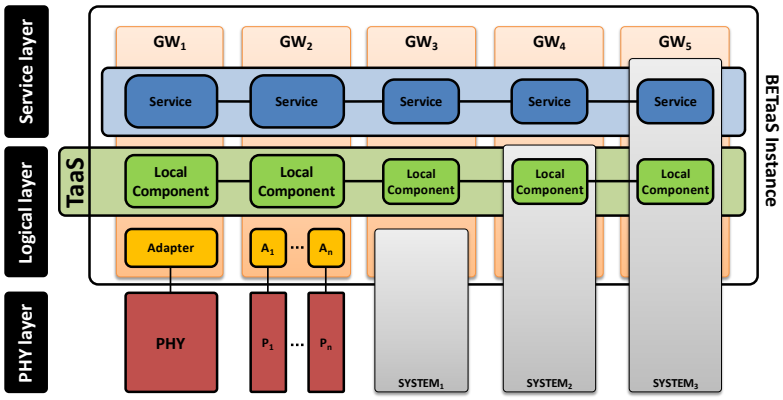

Figure 1: BETaaS functional model instance – mixed BETaaS-aware and BETaaS-unaware M2M systems

Service layer, the Thing as a Service layer, the Adaptation layer and the Physical layer, respectively. In order to emphasize the distributed nature of the BETaaS architecture, Figure 1 highlights the relationship between the functional layers, gateways, and IoT/M2M systems integrated into the platform.

The core of the platform is represented by the Adaptation and the TaaS layers, which enable integration and access to different existing IoT/M2M systems plugged into the platform through one of its gateways. Both of them together guarantee transparent access to physical objects regardless of their physical model or their location. Any physical system to be integrated into the platform must implement a minimum set of basic functionalities. This minimum set, described more in detail in the following, includes the basic functionalities that can not be implemented at the upper layers. Differences among the physical systems are conformed at the **Adaptation layer**, which exposes a

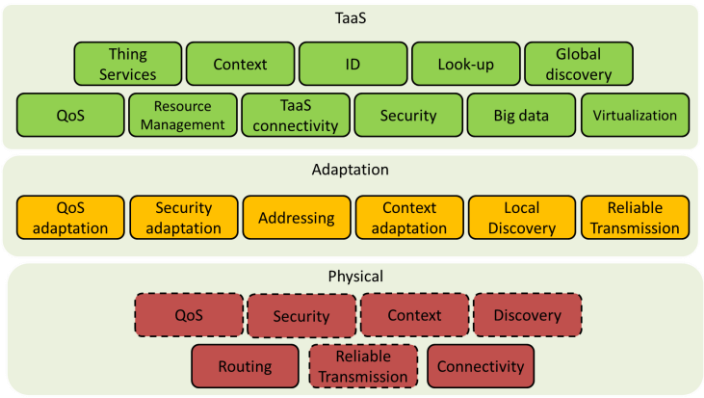

Figure 2: TaaS functional view.

common interface to the TaaS layer. This interface is deployed as a set of APIs, which are used by the TaaS layer to access the functionalities offered by IoT/M2M systems in a uniform manner.

The **TaaS layer**, by definition, enables the service layer to access things as a service. TaaS is implemented in a distributed manner: each gateway runs a TaaS local component, which connects to its peers to provide access to things regard-less of their location. A service requiring access to one thing interacts with its TaaS local component, which represents its unique interface towards the things. The local component is then responsible for accessing the thing through its own Adaptation layer, if the thing is connected to the local network, or for forwarding the service request to the TaaS local component of the gateway where the thing is connected. Instead of providing direct access to the uniform interface provided by the Adaptation layer, TaaS is included in the design not only to provide location-independent access but also to enrich data with context information in order to allow context-aware discovery and access.

The **Service Layer** is built on top of the TaaS layer: it provides services to applications leveraging on the things ac-cessed through the TaaS layer. A service can implement basic functionalities, which can be mapped directly to a single physical object, or can implement extended functionalities, which leverage on several basic services.

A distinction is made between BETaaS-aware and BETaaS-unaware IoT/M2M systems, respectively. The latter are fully unaware of the BETaaS platform, i.e., of its functionalities and interfaces (though could well implement inside some of these functionalities), and therefore need a dedicated Adaptation layer in order to be seamlessly integrated. This option allows for integrating any system into BETaaS, though at the cost of the possible replication of some functionalities. On the other hand, the former type of systems, i.e., BETaaS-aware, are integrated into the system by implementing some of the BETaaS layers: some or all of the BETaaS functionalities are implemented by the integrated system in a fully interoperable manner, i.e., compliant with BETaaS interfaces. In this manner, interoperability is ensured by definition while a higher level of efficiency in the implementation can be achieved.

The proposed reference architecture is represented by the functional view in Figure 2. *Resource management* is responsible for monitoring and controlling things in order to provide up to date information regarding the status of each thing, e.g., the status of the battery or the response time. *Security* is in charge of security, trust and privacy among gateways and with things. *Big data* manages the collection, storage and maintenance of data for statistical processing. *Context* manages the semantic description of the things, which is not limited to the type of data that a thing can provide but includes also the metadata parameter associated to each thing such as the geographic location.

Global discovery is used in order to retrieve the thing services available in a TaaS instance (exposed by other gateways), while *look-up* is used to search and resolve thing services based on context information. *QoS* provides functionalities used by applications to negotiate and request a certain QoS for a thing service. *TaaS connectivity* manages the communication between different TaaS local components inside the same TaaS instance. *Thing Service* is used to expose the things as services. *ID* is needed in order to globally address a thing among the entire TaaS instance. The id assigned to each thing service is called thingServiceID.

More details on the TaaS reference model and architecture can be found in [2] .

3.3. Semantic technologies and use of ontology

In BETaaS we use semantic technologies to: (i) integrate data coming from heterogeneous sources to provide a unified representation of them; (ii) discover information in heterogeneous and non-uniform environments; (iii) define a context-aware framework in which the circumstances that surround IoT elements are taken into account; (iv) map contents generated with the objects connected; and (v) manage the generation of knowledge from raw data. All these processes are performed by the Context Manager of the BETaaS platform.

To get these goals, we have created a unique networked ontology, the BETaaS ontology, which manages the knowledge of the things connected to the BETaaS platform and their context. The BETaaS ontology is built upon a network of ontologies which is created by reusing ontologies that are relevant in their domains and model the BETaaS scenarios. In particular, the following ontologies have been used: SSN4 (to model sensors and actuators), OWL-Time5 (to model temporal concepts), CF6 (for the representation of climatic and forecast data), Phenonet7 (to represent sensor and actuator types), MUO8 (to model the units of the measurements taken), FIPA9 (to model the capabilities of the things) and GeoNames10 (to model city names). Ontology development has been performed by means of Apache Jena11, which is an Open Source semantic web framework for Java.

When a new thing is connected to the BETaaS platform, all the information related to that thing is inserted in the BETaaS ontology. The process followed is summarized below (see Figure 3):

- Resources in the Physical Layer expose their data in different formats (e.g. XML, raw data).
- The Adaptation Layer takes the data provided by each of the things, and offers the user (e.g. the installer of the thing) the possibility to add contextual information to extend that data. A form is showed to the user with different parameters to be completed (e.g. location of the thing, type of thing, unit of its measurements, etc.) A set of parameters is created per thing.
- The set of parameters is translated into JSON format.
- The semantic parser translates the JSON file to an appropriate semantic format (RDF) and inserts the data into the ontology.

[4] http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628
[5] http://www.w3.org/2006/time
[6] http://www.w3.org/2005/Incubator/ssn/ssnx/cf/cf-property
[7] http://www.w3.org/2005/Incubator/ssn/ssnx/meteo/phenonet
[8] http://purl.oclc.org/NET/muo/muo
[9] http://www.fipa.org/specs/fipa00091/PC00091A.html
[10] http://www.geonames.org/ontology/ontology_v3.1.rdf
[11] https://jena.apache.org/

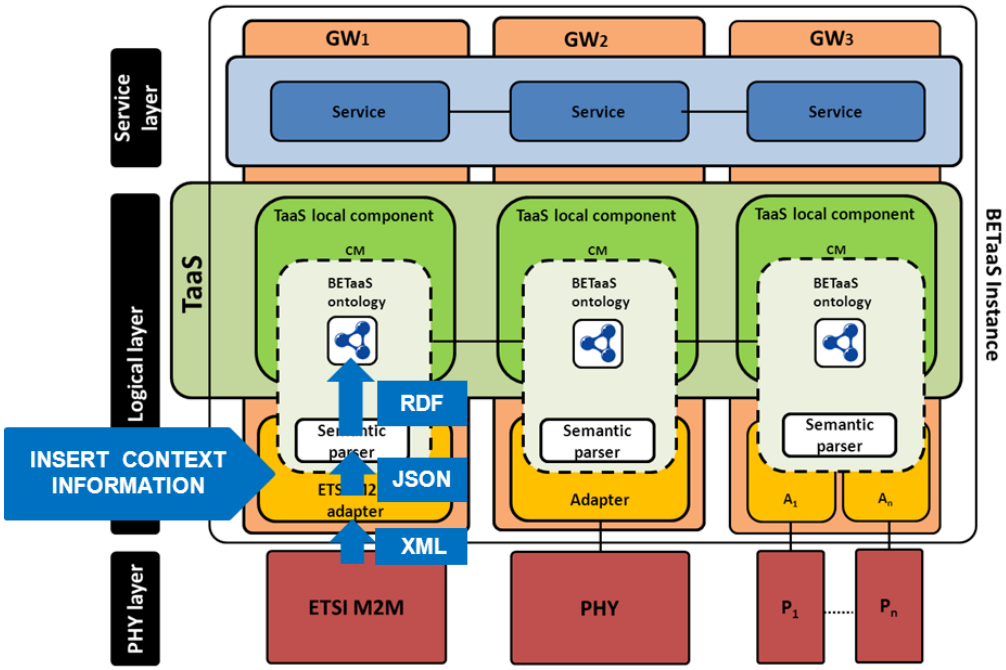

Figure 3: insertion of information

In order to promote standardization, the RDF information related to things is translated to WordNet synsets whenever possible, before storing it in the ontology. WordNet[12] is a lexical database that can be browsed online, that groups English words into sets of synonyms or synsets. These synsets are identified by a WordNet identifier. For words with different possible meanings represented by different WordNet synsets, disambiguation has to be performed. For example, the term *park* could describe a large area of land, but it could also describe a gear position. In order to perform disambiguation and select the correct meaning, a GUI is shown to the user with all of the definitions of the term.

WordNet organization is based on the semantic relationships between synsets (hypernymy, hyponymy, holonymy and meronymy). All synsets inserted in the BETaaS ontology are stored following these relationships, through SKOS[13], which offers a common data model based on RDF to organize classifications in a hierarchical way. The relationships between terms are used as a mechanism of knowledge inference. Inference can be applied at the time of the execution of applications: e.g. if an application demands the temperature at home, a temperature sensor installed in the kitchen is valid (*kitchen* is meronym of *home*). Inference can also be applied when registering thing types and thing locations in the BETaaS ontology: e.g. a new thing described as moistness sensor, would be added to a family in the ontology described as humidity sensors (*moistness* and *humidity* belong to the same WordNet family).

Among all the contextual parameters (introduced by the user) that describe a thing, there are two that are especially relevant: thing location and thing type. A location is described by means of two parameters: a WordNet keyword (*e.g. street, park, car…*), and a descriptive *free* text (*e.g. Oxford, Regency, John*). Location keywords are classified in SKOS hierarchies of related terms (e.g. *kitchen* has *room* as *hypernym* and *home* as *holonym*). In the same way, a thing type is described by means of

[12] https://wordnet.princeton.edu/
[13] http://www.w3.org/2004/02/skos/intro

a WordNet keyword. Similar thing types are classified in SKOS collections or families (e.g. *luminosity*, *brightness* or *light* belong to the same WordNet family). For both parameters, WordNet disambiguation should be performed in case it is needed.

These two contextual parameters allow the platform to create a *thing service* for each of the things connected to a gateway, following the nomenclature *setLocationType*/*getLocationType* (e.g. *getKitchenTemperature*).This nomenclature is extended with information about the gateway ID and the device ID (which is unique within a gateway): *getLocationType_deviceID_gwID (e.g. getMainBathroomHumidity_000003_000001).*

Apart from WordNet, we also use semantic rules to perform knowledge inference. More precisely, we have defined two semantic rules in the Context Manager of BETaaS, in order to: (i) detect equivalent Thing Services, which are those associated to things of the same type in the same location (*e.g.* two temperature sensors in the garden of a house would offer Equivalent Thing Services*);* (ii) detect the operator to be applied when different Thing Services have to be combined (e.g. if an application demands the presence at home, BETaaS would combine info from presence sensors installed in the kitchen, the bedroom and the toilet). Rules are executed by the Jena Rule Reasoner.

Finally, we have opened the information collected by the things connected to the BETaaS platform, implementing the Linked Open Data paradigm. Linked Open Data is based on four principles. BETaaS covers three of the four principles by means of the BETaaS ontology. To cover the fourth principle, it is needed to link BETaaS data to other linked RDF standards. To accomplish this, we have followed two different methodologies: (i) linking BETaaS data with a Linked Open Data dataset; (ii) publishing BETaaS data by means of a RDF vocabulary like DCAT, designed to facilitate interoperability between data catalogues published on the Web. Both methodologies are described following: on the one hand, by means of GeoNames, BETaaS is able to retrieve the name of the city in which a thing has been installed by using the GPS coordinates of the thing (automatically or manually defined) and reverse geocoding. To open the data, we have published this information by means of a combined solution that uses Apache Jena-Fuseki as a triple store, Eclipse Jetty[14] as web server and Pubby[15] as Open Source Linked Data frontend for SPARQL endpoints. On the other hand, DCAT methodology groups data on datasets described by a RDF standard. If requested by the instance owner, one dataset can be created per Thing Service. An instance can be seen as a catalog which contains N datasets. Each dataset contains information about a Thing Service and measurements taken. Each dataset stores the last 50 values measured by the corresponding Thing Service.

3.4. QoS, Security&Trust and Dependability built-in support

QoS support has been identified as a key non-functional requirement of the BETaaS platform. The ability of the platform to *provide different QoS guarantees to IoT/M2M applications accessing the Thing as a Service is of paramount importance in certain use cases*. Latency for example is a critical factor for applications such as real-time sensor monitoring in personal health-care or public safety systems. On the other hand, other applications like, e.g., road traffic management applications for urban mobility, though less sensitive to delay bounds, may nevertheless benefit from receiving some form of soft real-time treatment, at least for a subset of their provided services (e.g., 'urgent' alert notifications). Several other services like, e.g., power metering for accounting and monitoring at

[14] http://eclipse.org/jetty/
[15] http://wifo5-03.informatik.uni-mannheim.de/pubby/

home, are instead less sensitive to delay and could be assigned a lower priority to offset system and network workload congestion.

BETaaS has to provide QoS support to applications in order to support specific requirements of a wide range of M2M applications. The QoS functionalities offered by the platform are:

- **QoS negotiation**: applications can negotiate with BETaaS the desired QoS guarantee for a specific service.
- **Resource reservation**: BETaaS reserves resources in order to fulfill the committed guarantees.
- **Optimized allocation**: BETaaS performs allocation choices according to the characteristics and actual status of Things, e.g. computation capabilities and current energy consumption.
- **Different classes**: BETaaS provides to applications different service classes in order to offer a flexible and general QoS model.
- **QoS Monitoring**: BETaaS monitors the status of the resources available and the fulfillment of negotiated SLAs

Quality of Service (QoS) support provided by the BETaaS platform goes beyond the classic approach adopted in SOA architectures, i.e., QoS functionalities have to exploit the characteristics of the things and take into account the unique requirements of M2M applications.

Another important point identified in BETaaS is the need of trust models which may help to take better decisions about the thing services to use. Two models have been defined: one for evaluating thing services and another one for evaluating gateways. While the first one is used for avoiding the allocation of certain thing services (which are not behaving as expected), the second one is used for VMs allocation and for supporting the first model.

3.4.1. QoS Model

In order to guarantee that QoS is negotiated by applications in a uniform manner, a *QoS model is defined*. In order to reduce the complexity of platform management functionalities, a simple schema composed by three classes of services has been defined: Real-time, Assured, and Best Effort. The **Real-time** class is designed for applications with hard response time requirements where timing responses are usually mission-critical, e.g., surveillance alarm system, healthcare monitoring, industrial control. The negotiation phase is based on parameters, such as response time or service period, expressed in a deterministic manner. The platform must strictly comply with the QoS guarantees provided to this class of applications. The **Assured** service class instead is for applications with soft response time requirements. These applications usually tolerate some out-of-contract interaction; for this reason the negotiation procedure is based on probabilistic requirements. This class can be used by interactive application - ticketing or user information gathering – or may be used by tracking applications for logistic. Finally, the **Best Effort** class is used by applications that do not require any guarantee such as an application for historical data collection.

3.4.2. QoS Negotiation

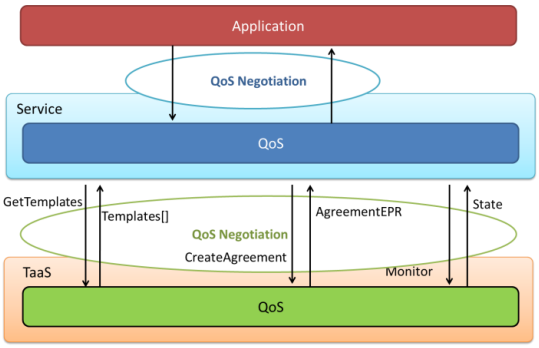

Figure 4: QoS Negotiation.

In order to allow the negotiation of QoS requirements between applications and the BETaaS platform, a standard service negotiation protocol is required. BETaaS adopts the de facto standard negotiation protocol called WS-Agreement [6] . The WS-Agreement standard is commonly adopted in order to define Service Level Agreements (SLAs) between providers and consumers. The WS-Agreement protocol defines the messages exchange by two end-points in order to create a service level agreement. The consumer asks to the provider all available templates. Then the consumer fills the template of interest and reply back to the service provider. Such interaction is called *CreateAgreement*. The service provider replies with a confirmation or a rejection message. Along with the confirmation message the consumer receives also an *Agreement End Point Reference* (AEPR) which uniquely identifies a created agreement. As can be seen in Figure 4, the WS-Agreement protocol is exploited between the Service and the TaaS Layer. In fact, to simplify the complexity of the implementation, applications specify the required services in a manifest that also contains the required QoS; the Service Layer, instead, exploits the WS-Agreement protocol in order to negotiate per Thing Service QoS requirements.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsag:Template wsag:TemplateId="1" xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement">
  <wsag:Name>BETaaS-Template</wsag:Name>
    <wsag:Context>
      <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
      <wsag:TemplateId>1</wsag:TemplateId>
      <wsag:TemplateName>BETaaS-Template</wsag:TemplateName>
    </wsag:Context>
    <wsag:Terms>
      <wsag:All>
        <wsag:ServiceDescriptionTerm wsag:Name="THING" wsag:ServiceName="THINGSERVICE">
          <betaas:ThingService xmlns:betaas="http://betaas.eu/schemas/betaas">
         <betaas:Definition>
                        <betaas:transactionID>$TRANSACTIONID</betaas:transactionID>
            </betaas:Definition>
            <betaas:QoS>
              <betaas:MaxResponseTime>$MAXRESPONSETIME</betaas:MaxResponseTime>
              <betaas:MinAvailability>$MINAVAILABILITY</betaas:MinAvailability>
              <betaas:MaxRate>$MAXRATE</betaas:MaxRate>
            </betaas:QoS>
          </betaas:ThingService>
        </wsag:ServiceDescriptionTerm>
      </wsag:All>
    </wsag:Terms>
</wsag:Template>
```

Figure 5: BETaaS Template.

The WS-Agreement protocol defines also the basic structure of XML-based templates and agreements. A specific schema aligned with the BETaaS QoS model and compliant with WS-Agreement has been specified. The schema describes the Service Description Terms required for the Thing Services invocation (an example instance is represented in Figure 5). The template follows the structure of the WS-Agreement standard: a *Context* section (<wsag:Context> tag) which contains the template name and the template id followed by a *Service Description Term* section (<wsag:ServiceDescriptionTerm> tag) which defines the terms of the Thing Service (<betaas:ThingService> tag). This section, in turn, includes the transaction ID needed to identify the Thing Service (<betaas:Definition> tag) and the QoS parameters, (<betaas:QoS> tag). In this example, a simple set of QoS parameters is included for illustration purposes:

- **MaxResponseTime**, used to specify the Response Time of the Thing Service. In detail, it indicates the maximum delay between a Thing Service invocation and its response, measured at the service layer.
- **MinAvailability**, used to specify the availability of the Thing Service. This parameter is associated to each Thing Service and is, in principle, a static parameter. However, since the environment can change unpredictably the QoSMonitoring functionality must control and adjust this parameter continuously.
- **MaxRate**, the maximum rate a Service can invoke the thing service, in other terms, a minimum inter-request time between two different requests from the same service.

3.4.3. QoS extended capability implementation

In order to enforce and monitor the negotiated QoS requirements, explicit support for QoS must be included in the platform architecture. Once the negotiation phase is performed, a SLA is established and applications could invoke thing services with the negotiated QoS level.

BETaaS QoS capability implementation relies on a two-phase procedure, namely, reservation and allocation. The reservation phase is handled by a sub-component called **QoSBroker**. The *QoSBroker* manages the QoS negotiation, performs admission control and, most importantly, manages resource reservation by exploiting equivalent thing services through a dedicated heuristic algorithm.

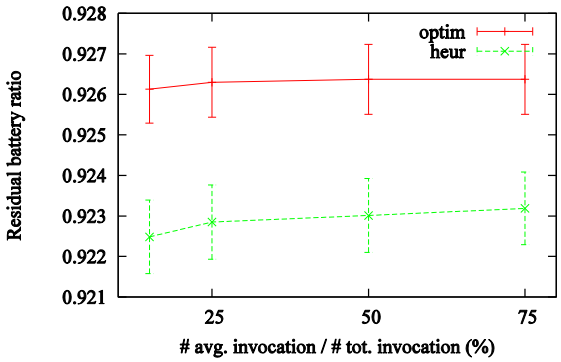

Figure 6: Min. residual battery ratio (50 things, 500 requests)

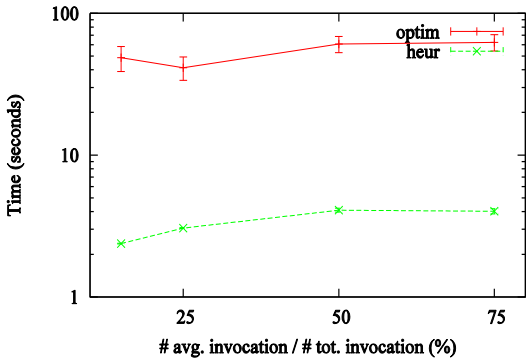

Figure 7: Computational Time (50 things, 500 requests)

It also generates AEPRs to authorize thing service invocation. The allocation phase, instead, is managed by another sub-component called **QoSDispatcher**. The *QoSDispatcher* performs allocation of resources at time of thing service invocation based on the reservation schema. The QoSDispatcher can handle things' failures and in the case trigger the reservation algorithm. Moreover, while the allocation procedure may be involved in each thing service invocation, the reservation procedure is executed only once at time of negotiation.

To avoid data inconsistencies, race conditions, and long response times, which can affect functionalities implemented in a distributed manner over large deployments, critical system functions, such as admission control, are provided through a centralized point of decision that is contacted whenever a decision cannot be taken locally. In the current release of the platform, for the sake of simplicity, it is assumed that this functionality is provided by a single gateway called GW*. To reflect this design, the QoSBroker and the QoSDispatcher are implemented by two sub-components with different scope: local and global, respectively. In the reservation phase, we rely on a **QoSGlobalBroker** and on a set of QoSLocalBrokers. The former is one for TaaS instance, while one instance of the latter is deployed in every gateway. The same approach is adopted for the QoSDispatcher (QoSGlobalDispatcher and a set of QoSLocalDispatchers). The local components manage, in term of QoS, the thing services provided by things that are directly connected to the gateway where the component is deployed. The global components, instead, have a global view of all the thing services available in the TaaS instance and are involved only when a global view is required. Further details on the interactions between local and global components can be found in [5] [6] .

A specific requirement of the BETaaS platform is the exploitation of the possibilities offered by equivalent things. Because of the integration of different systems, large IoT networks are characterized by a large number of equivalent things, which can potentially provide the same services. In this context, a *QoS framework that considers equivalent things by design can take full advantage of this variety to allow efficient management of resources*. Thing service equivalence has been exploited in the QoSBroker by implementing a reservation algorithm that allocate requests to things so as to maximize the lifetime of battery powered things (other optimization objectives are possible but were left for future development). The resulting optimization problem was formally

defined and proved to be an NP-Hard problem. A heuristic algorithm, named **Real Time Thing Allocation algorithm (RTTA)**, was therefore devised and implemented in order to find a solution close to the optimal one in a time suitable for implementation in a real system. Figure 6 and Figure 7 show the results obtained by evaluating RTTA in different scenarios characterized by a variable number of service requests and available things/thing services, as compared to the optimal solution derived by means of standard optimization solver, i.e., IBM ILOG CPLEX. In all considered case, it has been shown that the reservation computed by the heuristic algorithm is very close to the optimal one, but the time required for computation is in the order of a few seconds, more than one order less than the solver. Further details on RTTA and its evaluation can be found in [7] .

Once the *Real Time* service requests are allocated, the QoS algorithm must reserve resources for the *Assured Service* class, preserving at the same time the QoS requirements guaranteed to real-time services. The QoS Manager performs the overall allocation in two steps: first it allocates real-time requests to assure strict priority over other request classes, then it allocates the assured services. The allocation of assured services is performed through a modified version of the heuristic adopted for real-time services. Assured service requests are allocated using a *polling model*. The *assured service allocator* assigns each request to a specific PS based on QoS requirements and context information. For a detailed description, please refer to [5] .

The **QoSDispatcher** is in charge of dispatching thing service invocations to actual things based on reservations established by the QoSBroker. The Dispatcher algorithm is a greedy algorithm that first tries to apply the optimal selection, the thing selected by Broker. In case the optimal choice is not available at the time of invocation, the algorithm performs the best selection according to the actual system status. The latter is necessary to handle error situations due to system dynamics (e.g., temporary thing disconnection notified by QoS monitoring, or other system failures). For the sake of responsiveness, the dispatching algorithm considers only a single request at a time, without managing the whole set of requests that will be issued in a common hyper-period. Therefore, the actual allocation may be sub-optimal with respect to the one computed by the QoSBroker in case of transient changes of thing availability. If the latter becomes steady, a new reservation is triggered by the QoSDispatcher to the QoSBroker. Further details can be found in [5] .

Finally, a **QoS monitoring** functionality has been implemented aimed to achieve SLA monitoring and resources monitoring:

- **SLA monitoring**: it monitors Thing Services and it's in charge of maintaining the negotiated SLA agreements, taking appropriate actions to rectify non-compliance with a SLA. SLA negotiation takes place during the installation of a BETaaS application. The SLA monitoring of the Thing Services to be used by the application starts when an application starts its execution. These are the parameters that are checked to detect if a SLA violation occurs: (i) Availability of the thing service: if it is not available a SLA violation occurs; (ii) Response time of the thing service: if it is bigger that the theoretical response time (defined when connecting a new thing), a SLA violation occurs; (iii) Minimum inter request time: if time between petitions exceeds the default value, a SLA violation occurs. BETaaS performs SLA monitoring in real time, assuming that the SLA calculations are estimated as soon as the measurements from the things are obtained. At the same time, the SLA monitoring is performed after sending the response of the application, in order to avoid penalizing the application execution.
- **Resources monitoring**: it monitors things and it's in charge to ensure that the resources allocated by applications are available, notifying any particular change to the QoSDispatcher. The ideal design of this monitoring functionality was outlined in D2.2, defining several parameters to be monitored for all the things connected to a gateway (failure of a thing, reliability of the response time, percentage of reliability and response time). However, during the implementation and

integration of the different modules that compose the BETaaS platform, we had to tackle some performance issues that we managed to solve. In order not to penalize platform performance with a constantly running resources monitoring functionality, we decided to implement an "ad hoc" resources monitoring, diverted from the ideal one. So BETaaS platform implements a mechanism that monitors the resources that have been allocated, notifying the QoSDispatcher whenever a resource is disconnected from the platform. As soon as the QoSDispatcher receives this notification, it searches for an available equivalent thing in order to maintain service continuity.

3.4.4. Security and Trust extended capability

Security is a key enabler for M2M systems. Only if all stakeholders are able to establish a sufficient level of trust that their individual assets are sufficiently protected, they are willing to use M2M systems. In addition, in some domains like health or utilities there are also legal or regulatory requirements for the protection of M2M systems. Needless to say, the security holds a very important key to the wide acceptance of IoT/M2M systems, including the BETaaS platform.

The security built in support in BETaaS is provided by considering the unique feature of BETaaS platform, including the distributed cloud of gateways that forms a BETaaS instance. The security functionalities offered by the platform are:

- **Key management:** mechanism to manage and derive keys for authentication and encryption/decryption.
- **Authentication:** including the authentication between gateways, and between applications and the platform.
- **Authorization:** authorize application or user to access services provided by the platform.
- **Trust:** represents the level of reliability on a Thing or Thing Service to produce certain data and/or perform certain actions.

3.4.4.1. Key management

Key management is a very important functionality which defines how secret (and shared) keys, which are the important components to perform any security operation, are managed. The key management has a role to manage the associations of different entities (e.g. gateways with an instance, application with an instance, etc), which will be used to perform authentication, and later on to perform encrypted communication.

The key management aspect in BETaaS is focused on the gateway and instance level, mainly due to a minimum level of user intervention. As BETaaS instance consists of multiple distributed gateways, we need to make sure that secure communication and information sharing are in place. Secure communication means that the communication channel is encrypted with a certain key which is obtained in the authentication process. The authentication process itself can succeed when there is some level of security trust between the entities, which is usually relying on the public key infrastructure (PKI) in a form of digital certificate issued by a Certificate Authority (CA).

The key management in BETaaS requires local certificates issued by the "internal" CA, instead of global certificates issued by trusted third party. This "internal" CA refers to a gateway which acts as a common CA of that particular instance which practically resides in the GW* of a BETaaS instance.

At the beginning of BETaaS instance creation, the GW* would initiate the creation of root and intermediate certificates which forms the root credential of this particular instance. For a new GW to join the instance, it needs to generate a key pair, i.e. public and private keys, and then sends a join request to GW* along with the generated public key. Upon receiving join request, GW* would create a credential for the new GW which includes a certificate generated from the public key and other submitted information by the new GW and is signed using the private key of the GW*'s credential. This new created credential plays an important role to provide trust relationship among all the GWs within the instance, including the access control mechanism (i.e. authorization) as well as in deriving key for establishing secure communication between two GWs.

3.4.4.2. Authentication

We consider the authentication within in two cases: gateway level authentication (e.g. among gateways within an instance), and application and service level authentication (e.g. application that consumes services provided by a BETaaS instance).

The gateway level authentication mainly deals with the mutual authentication in order to establish secure communication between two GWs. The method to perform mutual authentication is based on the Elliptic Curve Menezes-Qu-Vanstone (ECMQV) protocol. ECMQV is a three-pass key agreement protocol that has been standardized in ANSI X9.63, IEEE 1363, ZigBee Smart Energy 1.0, etc. In principle, it is an extension of the ordinary Diffie-Hellman key agreement protocol with the ECC. Among the benefits of ECMQV are lower computational complexity and bandwidth reduction.

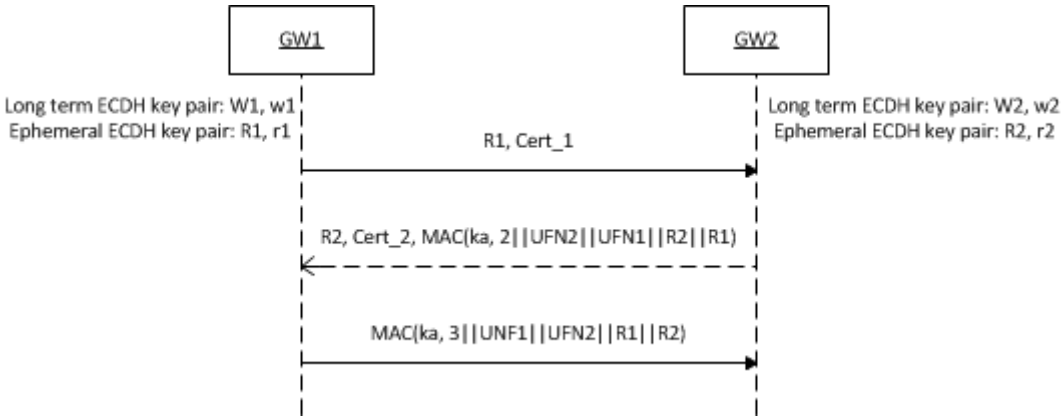

Figure 8: ECMQV key agreement protocol

Figure 8 shows the ECMQV key agreement protocol sequence diagram between GW1 and GW2 (GW1 is the initiator), where:

- W1 (W2): Long term public key of GW1 (GW2)
- w1 (w2): Long term private key of GW1 (GW2)
- R1 (R2): Ephemeral public key of GW1 (GW2)
- r1 (r2): Ephemeral private key of GW1 (GW2)
- Cert_1 (Cert_2): instance certificate of GW1 (GW2)

- MAC: Message Authentication Code
- ka: one part of the derived secret key between GW1 and GW2
- UFN1 (UFN2): User Friendly Name of GW1 (GW2)

The purpose of this key agreement protocol is to derive a key (let us call it k_b – the other half of k_a), as a secret key that will be used to authenticate GW1 and GW2 and to establish encrypted communication between GW1 and GW2. Both k_b and k_a are calculated independently in each of the GW based on the GW's own key pairs and the public keys from other GW: R1(2) and W1(2) (where W1(2) is extracted from the Cert_1(2)). After deriving the keys, both GWs will send a MAC which contains concatenated strings of 2, UFN2, UFN1, R2, and R1 for the first MAC, and 3, UFN1, UFN2, R1, and R2 for the second MAC, and encrypted by k_a. If both GWs can verify the MAC, it means that they have derived a valid secret key k_b (and k_a), thus achieves a mutual authentication. More detailed explanation of how ECMQV protocol works can be reviewed in [8] . Afterwards, the derived key k_b is used to encrypt/decrypt the data being exchanged by GW1 and GW2 by using AES encryption algorithm.

With regards to the application level authentication, it is performed to check the authenticity of the application at the installation or registration stage. As a side note, an application needs to perform an installation or registration procedure in a BETaaS instance before it can actually access the services provided by that particular instance in order to check the availability of the resources and to allocate it according to the requirements set by the application. In order to authenticate the application, it is also based on the certificate issued by the trusted CAs by the BETaaS. The certificate can also be generated by the BETaaS CA and obtained by the application developer prior to developing an application for BETaaS, i.e. through a registration procedure.

3.4.4.3. Authorization

In BETaaS, the capability based access control approach is used by considering some relevant advantages, such as scalability, least privilege access principle, and flexible yet controlled access rights delegation. The overall design along with the mechanisms involved in the proposed capability based access control for BETaaS platform will be presented in the following subsections.

Capability design

The proposed capability consists of internal and external capability, e.g. $_{in}CAP$ and $_{ext}CAP$ respectively. In BETaaS's context, the resource or object is the *thing service* which is owned by a GW, thus *inCAP* is created and owned by any GW that offers *thing services*. *Thing service* refers to a service provided by a *thing*, e.g. get temperature by a temperature sensor. The *extCAP* can be obtained by any party that wishes to access a resource that corresponds to the *inCAP*, including another GW in the instance, applications, and users.

The basic structure of $_{in}CAP$ is as follows:

$$_{in}CAP = \{O, DSign\}$$

where

- *O*: The name of object or resource to be accessed.

- *DSign*: A digital signature of this $_{in}CAP$, which is obtained by signing the hash of the $_{in}CAP$'s content, i.e. *O*, with GW's private key that is obtained by the key management mechanism in Section 3.4.4.1. It is used to check the integrity of the capability's content.

On the other hand, $_{ext}CAP$ contains much more information than the $_{in}CAP$ which is necessary for the capability delegation and revocation mechanisms.

$_{ext}CAP = \{I, S, AR, O, VC, IC, R, DSign\}$

where:

- *I*: The information about the *issuer* of this $_{ext}CAP$.
- *S*: The information about the *subject* or the $_{ext}CAP$ holder.
- *AR*: The details of *access rights* over the object or resource to be accessed.
- *O*: The name of object or resource to be accessed.
- *VC*: The *validity condition* of the $_{ext}CAP$, e.g. validity period.
- *R*: The URL of the capability *revocation* service.
- *DSign*: A digital signature of this $_{ext}CAP$, which is obtained by signing the hash of the $_{ext}CAP$'s content with issuer's private key.

Capability creation mechanism

Initially, the object or resource owner, i.e. GW, creates $_{in}CAP$s for all the objects that it has authority over. In BETaaS context, an $_{in}CAP$ is created when a *thing* that provides *thing services* is connected to the GW. The $_{ext}CAP$ can then be granted upon request by the external parties that wish to gain access to a resource that coresponds to that particular $_{ext}CAP$.

Access right delegation mechanism

The access right can be easily delegated by propagating the *extCAP*, but it is difficult to control its propagation. Some attempts have been made to control the capability delegation in the previous works. Our previous work uses a mechanism where the delegator needs to ask the original resource owner before delegating its *extCAP* which reduces the flexibility of the access rights delegation through capability. On the other hand, there was also an attempt to control the right delegation by specifying the delegation depth in the access right field within the capability. However, it is impractical to predict the proper depth of delegation in advance as one can always need to go one more level down.

In the proposed approach, it is assumed that the $_{ext}CAP$ is delegable by default, i.e. no need to specify the *delegable* and *delegation depth*. Anybody can delegate a subset of its access rights stated in the $_{ext}CAP$ by creating another $_{ext}CAP$, specifying its ID, certificate, and a list of $_{ext}CAP$ chain in the issuer field, and sign the $_{ext}CAP$ with its own private key. When the delegatee, i.e. the $_{ext}CAP$ holder

that receives a delegated access, accesses the resource and presents its $_{ext}CAP$ to resource owner for the first time, the resource owner will evaluate the $_{ext}CAP$. The most critical point in the evaluation that controls the access right delegation is by validating the $_{ext}CAP$ issuer based on the issuer's certificate (for GWs and applications) or user credential (for the registered user in the corresponding GW). If the certificate or credential cannot be validated, it simply means that the issuer has no right to delegate the $_{ext}CAP$, thus access request is rejected immediately. Otherwise, the $_{ext}CAP$'s signature is validated based on issuer's public key to ensure that the delegated *extCAP* has not been tampered either by the delegatee or other parties. Upon successful delegation, a new $_{ext}CAP$ signed by the resource owner's private key will be granted to the delegatee, thus allowing a simpler validity check as a normal $_{ext}CAP$ in the next access time.

Capability access evaluation

Upon access request by receiving the $_{ext}CAP$, the resource owner validates the submitted $_{ext}CAP$ by checking its validity period, validating the $_{ext}CAP$'s *DSign*, and definitely checking whether the requested access right is listed in the $_{ext}CAP$. In addition, the resource owner also checks the validity of the $_{in}CAP$ and/or $_{ext}CAP$ which are specified in the *IssuerCapabilities* field, and whether or not the $_{ext}CAP$ has been revoked through the revocation service. To make it even more secure, the $_{ext}CAP$ holder can also sign the $_{ext}CAP$ and submit it along with its public key information to prove that he/she really is the $_{ext}CAP$ holder, e.g. in case the $_{ext}CAP$ is stolen.More detailed explanation of Capability based Access Control in BETaaS can be reviewed in [9] .

3.4.5. Trust Models

Two trust models have been defined with different purposes. The first one evaluates provision trust of thing services, in order to determine whether they are able to provide the expected data in the expected conditions. For doing so, the model evaluates six aspects: Security Mechanisms (check whether it is possible to use encrypted communication channels), QoS Fulfilment (determine whether SLAs are violated or not when using a concrete thing service), Dependability Performance (determine availability issues related to a thing service), Performance Scalability (check whether the thing service still responds in the expected time even when requests increase), Battery Load (calculate if the thing service will be available according to battery levels) and Stability in Provided Data (determine whether the generated data is coherent).

Several calculations have been defined for evaluating the mentioned aspects. In several aspects (Security Mechanisms and Stability in Provided Data), fuzzy models have been used for aggregating certain properties and they have also been applied to aggregate all the aspects evaluation, as a mean to get a global trust value. Opinion models have been applied to QoS Fulfilment and they were complemented with some statistical tests, which were also used in the Stability in Provided Data and Dependability Performance aspects. Linear regression is used for determining the slope representing the Performance Scalability. Finally, double exponential smoothing was applied to Battery Load, as a way to forecast battery levels.

The second trust model is about evaluating gateways and their capability to provide functionalities to the BETaaS Instance in which they are participating. In this case, five aspects are evaluated: Historical Interactions (how the gateway behaved in previous cases), Gateway Dependability (determine issues with the components working in the gateway), Path to the Gateway (how far is in the network and which nodes are in the path), Gateway Energy (availability of the gateway due to energy levels) and Gateway Reputation (opinion of the rest of gateways about a concrete gateway).

Opinion models are used again, but this time for Historical Interactions, with some changes in the formulas. Double exponential smoothing is also used in the Gateway Energy aspect, since it is quite similar to the calculation done for thing services. For evaluating Path to the Gateway, a Dijkstra's algorithm is used on the graph representing the network. In the case of Gateway Dependability, we calculate a survival function based on the Kaplan-Meier estimator. Finally, Gateway Reputation is determined by applying weighted averages controlled with a Fleiss' Kappa coefficient. As in the case of thing services, all the aspects evaluations are aggregated by using a fuzzy model, as a way to get a global trust value.

3.4.6. Dependability built-in support

The BETaaS platform leverages an information queuing system to manage the processing information coming from the different software modules. In particular different queues are used to route different kind of information. Such data can be inspected by the platform administrator through a Web GUI in order to take decisions about the system or get acquainted about critical elements or failures.

Since the platform is based on Distributed OSGi, the dynamic bundle deployment capability is exploited to make up an automatic software modules recovery, in case some of them should stop working properly and become unresponsive.

3.5. Validation: BETaaS in the real scenarios

In our era, many aspects of everyday life are dictated by how well electronic equipment or machines controlled by such equipment operates. It is most likely that the next level of this relation between humans and computers will be based on how well is the communication and cooperation between machines themselves. Furthermore, there is the growing need to organise the ever increasing amount of information coming from the ever advancing and multiplying hordes of digital sensors spread throughout the physical world.

BETaaS primary target is to have this information managed and organised in a useful manner for the everyday user. Eventually the user is allowed to exploit and benefit from the transformation of simple devices into meaningful services provided to specially built applications (or BETaaS supportive applications). Thus we move from a simple sensor/actuator to a fully operating and co-operating service for that device that provides more than just a reading, such as for example qualitative measurements by means of not providing a measurement if the device is deemed unreliable or untrustworthy. Additionally, different kinds of devices (i.e. communication protocols) usually require their own particular closely coupled software to operate. BETaaS solves this issue by easily incorporating software adapters for different M2M (Machine-to-Machine) communication protocols.

Towards this target two major demonstration stories are told here, highlighting the major points that BETaaS platform wants to convey to the users. These stories refer to the Home Automation and the Smart Cityenvironments.

3.5.1. Home Automation

Imagine George, a person quite familiar with the technology surrounding devices such as sensors and automatic switches. He does not need to know the very details behind electronics but is well accustomed to the networking and setting up of the devices surrounding him at his very own house. He is the owner a proprietary software system that acts as a domotics/automation system providing

assistance in everyday activities and also some basic security, since it monitors temperature, doors closing or opening and presence through appropriate sensors. He finds out about a new platform, BETaaS and a particular BETaaS application for organising and running the watering of his garden. George only needs to patch his existing system, download the application from the marketplace available and install a humidity and watering valve. The application guides him through the various steps needed since it can scan the systems it connects to and any missing service is translated to a physical device that was not discovered and must be installed. Once the humidity sensor and the switch for the water are installed the application informs George that all the required devices and therefore services are now active and working and that the scheduling of watering cycles can be organised according to his input.

It is important to highlight the fact that the devices can belong to different makers, as long as they are all supported by the current version of BETaaS. Additionally, in case a sensor or switch is deemed improper by the heart of the BETaaS system it will be left out and its service should be recruited from another similar sensor if possible. This is how BETaaS manages to relief George from the bulk and erroneous procedures of installation and wiring of the devices' information all the way up to an application that can reasonably use it.

3.5.2. Smart City

The Smart City scenario has been designed to specifically focus on the following platform features:

- QoS
- SLA monitoring
- Big data
- Scalability
- Ontology

Pisa is a city in Tuscany, a region in central Italy.

It's a small city (only 89,000 residents) but everyday a great number of persons move to the centre for its touristic attractions (the leaning town for examples) and its offices (e.g. university and hospital).

Traffic, pollution and parking is a great problem for Pisa; in order to solve these problems, municipality decided to control its own parking using traffic and pollution information to open or close the service, encouraging people to leave their cars in the peripheral areas, when needed.

Municipality decided to use BETaaS to control the parking service to ease the integration of traffic sensors, pollution sensor that were already installed but used for other services.[**Scalability**]

New sensors were installed to control the access to the parking (remote controlled semaphore), to control the entrance and exit of the cars (entrance/exit sensors) and tuneable lamps to reduce the waste of energy when the parking is closed.

Pisa has chosen ETSI compliant sensors because ETSI is a standardization organization to which a great number of ICT and M2M devices producer are associated.

So the municipality has decided to apply the following rules:

- if traffic is intense near a parking, the parking is opened, the semaphore at the entrance is turned to green and its lamp posts are switched on.
- if high pollution is detected in the city centre, then peripheral parking will open and central ones will close (controlling lights and semaphores accordingly). When a parking closes, lights are turned off only after no car is inside.

8.00 AM Carlo 21 years old is a student of Engineering faculty of Pisa University, he is approaching to the centre of Pisa by his car. The traffic is not much intense in the peripheral, but in the centre it's getting intense, so the parking in the centre of Pisa are opened. Carlo checks on his smartphone which parking are open, he uses a dedicated App: "Pisa parking app".

9.00 AM the traffic is heavy in the centre of Pisa, so the pollution reaches a dangerous level, the parking in the centre will be closed and the parking in the peripheral will be opened. Maria 50 years old works in a bank in the centre of Pisa, she checks the Pisa parking App, she is angry because she is late and all the parking in the downtown are closed, she approaches to the peripheral parking that is nearest to her.

12:00 PM Paolo, 55 years old, is ICT technician at the local police force. From the control GUI (a web user interface provided by BETaaS platform) he checks the statistics of the traffic and the openings/closures of the parking so that he can plans the positioning of the police men for the next day. [**Big data**]

5:00 PM Fabio, 40 years old, is a doctor at Pisa hospital, he checks on his smartphone the pollution level in the city of Pisa. He uses an App that allows the user to check the pollution level of the selected city; it research in the BETaaS instances of Italian municipalities. [**Ontology**]

8:00 PM Giulia 38 years old went to the centre of Pisa to do shopping, she left her car in the peripheral parking. She takes the bus to go back to her car, the only one still in the parking. The traffic in peripheral of Pisa is light so the parking will be closed, but Maria is still there, so the lights in the parking will not turn off until she goes away.

9:00 PM Sara 25 years old, is the ICT technician for the Pisa municipality, she checks the status of the sensors: she checks that in the "Barbaricina" area at 10:00 AM, one of the 3 traffic sensors (Traffic1) stopped working, so an equivalent one (Traffic2) was automatically used by the system to replace Traffic1. At 7:00 PM Traffic2 stopped working, but Traffic3 wasn't used by the system because it couldn't guarantee the requested Quality of service. [**QoS**] [**SLA monitoring**]

4. Potential impact, main dissemination activities and exploitation of results

4.1. Contribution to the State-of-the-Art

Today there are countless devices at work to improve productivity and quality of life of human beings, in all technological domains. In most cases they operate in isolation or with very little cooperation from their likes, and serve a well-defined single purpose for which they have been engineered. Due to the recent advances of device manufacturing and communication technologies, new devices with Internet connection are being put forward and their penetration is expected to grow exponentially in the next years. This will provide an unprecedented opportunity to create applications and services that go far beyond the mere purpose of each participant, as enabled by the interactions in this so-called Internet of Things (IoT). In principle, all the things of the world can be empowered to become "smart" and communicate with other peers or remote systems to better serve their purpose. A common aspect of IoT applications is that many of them of practical interest involve control and monitoring functions, where human-in-the-loop actions are not required. As a matter of fact, the only reason for having many of these applications is to remove human intervention for improved efficiency, security, and safety. In BETaaS project we focused specifically on these applications, which we call Machine-to-Machine (M2M), which will create a bridge between the real world (made of sensors, actuators, tags that are pervasive in our lives) and the virtual world (the Internet and its associated services).

In BETaaS project we started with an in-depth analysis of the state-of-the-art in the major areas of the project, i.e. Dependability, Security, QoS, Content-centric M2M applications, M2M architectures, M2M basic platform capabilities, Virtualization environments, Run-time environments and Big Data. In addition to that we investigated key running and finished EC projects; comparing them in terms of overlapping areas and areas that we differentiate. We have also followed standardization bodies and fora that were relevant with BETaaS. All the above defined the baseline for our activities. BETaaS contributed to the State-of-the-art by progressing its status and addressing drawbacks. The main areas of contribution are listed in section 4.2.

4.2. Scientific Impact

The characteristics that the project has exploited and further developed, achieving a scientific impact, are listed below:

4.2.1. Dependability

Within the dependability research area, a common approach leading to an end-to-end solution was missing; by considering the wide aspects of dependability, specific and ad-hoc architectural and easy-to-implement solutions have been followed in BETaaS. Moreover, also within the industrial area, even though the dependability represents a priority driver, it is described in terms of general objectives and not in detail; the ETSI M2M standard could be mentioned as an example. Nevertheless, some concepts and main solutions were considered as common references and objectives. In particular self-healing, diagnosis and recovery actions play a key role. These topics represented the leading factors for BETaaS and they were reached mainly by the exploitation of the OSGI framework. The collaborative aspect of the framework allows the platform to self-identify failures and unexpected behaviours; furthermore one of the main capability is to offer a high level of isolation between the SW components and modules guaranteeing service reliability and continuity. Moreover the Virtualization capabilities offers a further level of isolation.

4.2.2. Security

Existing trust models are not so oriented to the Internet of Things environment and, in those cases they can be applied, they are more focused on the analysis of the networks crossed by the data received. Even if some aspects for doing calculations can be used as inputs, BETaaS defined its own model, based on the things represented by thing services, but oriented to represent the trust of the thing service, which is the entity exposed to the applications.

4.2.3. Quality of Service

BETaaS adopted the WS-Agreement protocol, which is the de facto standard for SLA negotiation WS-Agreement, defines a language and a protocol for advertising the capabilities of service providers and creating agreements based on templates, and for monitoring agreement compliance at runtime. An agreement between a service consumer and a service provider specifies one or more service level objectives both as expressions of requirements of the service consumer and assurances by the service provider on the availability of resources and/or on service qualities. Moreover BETaaS exploited the QoS functionalities provided by different systems in order to uniform the QoS support offered to the overall BETaaS system. The adaptation layer was in charge of providing a uniform interface defining which features re-implement, which features adapt and which features use as they are.

4.2.4. Semantic Sensor Web Technologies

In this area, the main innovation is not related to the used of specific innovative technologies, but to the use of semantic sensor web technologies applied to an M2M environment. Nowadays, M2M applications are closed and proprietary systems, which do not allow re-use across different applications. Besides, most of the M2M solutions are designed from the device or the service point of view, forgetting to set the target on the content exchanged. In order to solve these issues, BETaaS proposed the use of semantic technologies, particularly those relating to the Sensor Web, which are the best suited for modelling M2M environments. The use of semantics allowed BETaaS to map content information with thing resources in a context-aware fashion. This helped BETaaS to dynamically adapt to changing environments, one of the main problems of M2M networks. Also, by modelling the behavior of the things BETaaS is able to automatically react to unexpected circumstances or to changing conditions, in the same way that a human would do.

4.2.5. Linked Open Data

The opening of the data can be seeing as a fuel for innovation. Interlinking the data from the physical world and the Web complements one of the key potentials of semantic Web to create a networked knowledge infrastructure. The publication of the measurements taken by the things connected to BETaaS in the Linked Open Data Cloud, opens the possibility of linking the information managed by the things with the one existing in the Internet. This allows the use of M2M data in innovative ways, for example by suggesting relevant sensors based on their location. In general, the use of Linked Open Data allows further exploitation of the information retrieved by M2M solutions, by any third party that develops Big Data applications.

4.2.6. Discovery, brokerage, and heterogeneous standards

BETaaS uses OSGi, particularly the Distributed OSGi (DOSGi) feature supported by Apache Karaf, i.e. an OSGi runtime environment implementation, to perform service discovery functionality. Services in BETaaS were implemented within the logical gateway and so are the interfaces or APIs. In a distributed gateways environment, as promoted in BETaaS, discovering relevant and available

services in different gateways is a challenging task. To cope with this, DOSGi and Apache Zookeeper, another feature of Apache Karaf, are utilized to allow dynamic bundles and services discovery. In this approach, the Zookeeper server registers all the DOSGi bundles, thus the gateways do not need to know each other IP address, they only need to know the address of the Zookeeper server. Moreover, the services in DOSGi are exposed as web service interfaces, which provide wide range of interoperability with heterogeneous system.

4.2.7. M2M architectures

One of the key features of the BETaas platform is the ability to potentially interact with any kind of sensors. The platform architecture easily allows adding one plugin for each technology to take care of the corresponding communication protocols. The goal was to make BETaaS solve the main problem of current M2M solutions that is the fragmentation of standards and technologies. Besides allowing the use of many device technologies through plugins, the project specifically focuses on the ETSI M2M standard that is going to be a promising solution to next M2M systems. By just plugging-in the ETSI plugin into BETaaS, the platform opened to this class of sensor networks that is likely to soon become the most widespread M2M standard.

4.2.8. IoT middleware solutions

BETaaS integrated existing technology, such as GSN, through adaptation layers. In particular, GSN gathers data from sensors by means of data streams. Each data stream can be mapped to a thing in a trivial way. The adaptation layer for GSN was deployed exploiting the unique features of GSN, while the missing functionalities were entirely implemented inside the BETaaS environment.

4.2.9. M2M basic platform capabilities

The access technologies to close elements, particularly the PHY and MAC layers of those technologies, were not the main concern in BETaaS development, but concern more on the much higher layer protocols, like CoAP, that at the end will (theoretically) be working in devices implementing various access technologies.

Regarding the access technologies to make service request, BETaaS supports both RESTful and SOAP interfaces. The SOAP (XML) web service interface is particularly used in GW to GW communication, using DOSGi as the underlying technology. The RESTful interface or API is particularly more convenient for the application to access services provided by BETaaS platform.

4.2.10. Virtualization environments

With respect to virtualization environments, BETaaS used state of the art hypervisors, such as the last version of Xen, which now include support for devices with ARM processors (those providing native virtualization operations). BETaaS built a layer on top of the hypervisor in order to manage VMs locally and it will decide when and how to deploy VMs, something which cannot be done in other IoT-based platforms. Other hypervisors such as KVM are not so mature, but they could be adopted in the future, since the VMs management layer will be hypervisor agnostic.

Moreover, BETaaS exploited current Cloud interaction standards in order to manage resources in remote Clouds, when resources available in the local cloud are not enough. It will be possible to interact with IaaS platforms such as OpenStack, OpenNebula, etc.

4.2.11. Run-time environments

The high level of modularity required by BETaaS components has been achieved through OSGi. In fact this specification allows deploying at run time new bundles without affecting the whole runtime environment. OSGi itself has been around since long time, but recently a new family of containers have been developed to extend the features offered by this framework. These containers support one or more OSGi implementation and allow to easily deploy new components by the mean of a CLI (command line interface). For BETaaS, the chosen container has been Karaf. The reason is that it supports the deployment of features like DOSGi natively, it is actively maintained and evolved and also it provides specific features that enhance the runtime environment: the so called features support. A Karaf feature is a logical group made of bundles that can be deployed as a whole. For example a feature defines bundles that need to be deployed in the container to support database at TaaS layer. In this way BETaaS gateways were deployed with specific features tailored for their capabilities and purposes. Also the features were used to define requirements in terms of dependency, so a feature is deployed after its dependencies are installed. Another important support that OSGi runtime environment takes from Karaf capabilities is the support of configuration files; these files provides bundles with environment specific settings, called properties, so that each one could be deployed with specific settings.

4.2.12. Big Data

As the number of gateways connected in a BETaaS instance will grow, the problem of keeping these large and growing amount of data arises. For this reason BETaaS defined a set of technologies that are used with the purpose of storing, manipulating and processing information. The used technologies are mostly based on the Hadoop ecosystem and specifically based on the Cloud era Distribution of Hadoop (CDH). The reason behind this is that it is easy to install and configure beyond the Hadoop itself: in fact it can be used to easily deploy and manage Big Data components through packages, that are also used inside BETaaS. The following components were considered in BETaaS: HDFS, Hadoop YARN, Apache Sqoop, Apache Hive, PrestoDb.

HDFS is a distributed file system (Hadoop Distributed File System) that is going to be used to store the large amount of data collected by gateways. The data can directly be stored to HDFS or collected from databases and loaded on HDFS. For this last purpose it will be used Apache Sqoop: it consists in a server component that can schedule batch job for loading SQL data into HDFS files and vice versa. The other technology that we are going to use is Apache Hive. This is a datawharehouse tool that allows defining tables over HDFS files. While it will not be directely used by BETaaS application, the reason for using Apache Hive is that it allows defining tables that PrestoBD can query through its SQL language. In this way data taken from different sources (BETaaS gateways and databases) are collected on files stored by an HDFS infrastructure. This information then is queried through an interface based on SQL offered by Prestodb. In this way, while support for large amount of data is guaranteed to BETaaS by HDFS and the Prestodb architecture, application can query and process information by using standard SQL interfaces. For more complex data processing jobs, which for example need to perform processing of whole historical data, BETaaS offered specific Hadoop map/reduce jobs that can be deployed inside the instance.

The innovative approach of BETaaS is the delivery of standard patterns of information processing based on traditional SQL languages also over its Big Data capabilities by combining components that will cooperate to achieve these results. Moreover another innovative element of BETaaS is that these components will be deployed in an OSGi environment and over a very heterogeneous and distributed architecture like the one defined by the BETaaS gateways. Finally, BETaaS combined the real time analytics with the job batch processing offered by the Hadoop architecture.

4.3. Societal and Economical Impact

4.3.1. Economical

BETaaS approach for creating impact lies in the value of facilitating the way technology can improve ordinary people's life and business needs by empowering developers and companies to create IoT application and services: Enabling the Programmable World of Things with its outcomes and the Platform and Simulator released as open source.

Therefore, BETaaS creates impact in the economical dimension with its benefits and value for IoT and M2M. Some of its main benefits for the economy are based on its potential to:

- increase competitiveness (e.g. using open source technologies to increase opportunities to reuse and benefit from the ecosystem and by being able to develop new services and experiences with enabled things and applications for the IoT);

- reduce cost and time to market (e.g. reutilization of existing services to integrate new things by only creating an adapter for that type of thing, if needed, or by integrating vertical systems, until now hard to connect and force to collaborate together; or by the possibility of rapidly create applications using Things as a Service, instead of vertical and specific approaches; etc.).

- lower barriers (e.g. by abstracting underlying complexity and compatibility issues of heterogeneous technologies and multiple vendors; and with BETaaS high level capabilities and services that provide operation in cloud or "local-cloud" models, abstraction of Things world, context aware, semantic, big data, built-in QoS, Security and dependability etc.);

- improve and simplify software development process allowing the creations of IoT and M2M ready applications by a larger community (e.g. allowing the creation and testing of IoT applications without the need to physically deploy things in the real world with the use of BETaaS simulator tool). The release of BETaaS platform as open source benefits the IT industry in general, allowing others to reuse and extend technologies to create new solutions or business models and by reducing development costs and time-to-market. Also in terms of the benefit generated from contributions of the open source community of developers (in terms of customization, testing, developments, new enhancements, etc.). It also allows service providers to focus on the application-specific aspects, without the need for working on common features.

- use and integrate todays physical infrastructures to meet future demands and futuristic visions (e.g. for Smart Cities and Smart Home). This makes enabling the creation of "broader applications" using different underlying technologies seamlessly, abstracting underlying complexity of heterogeneous things a real problem that is being faced by different stakeholders. BETaaS with its Things as a Service approach simplifies the creation of things environments taking advantage of the horizontal approach for interoperability. All this translates into an existing opportunity and a challenge to create profitable business models while enhancing our lives.

- the endless possibilities that brings for companies to create applications, services, and experiences enabling co-creation of value. (e.g. BETaaS + UDOO = create your own IoT

app). The evolution and expected growth in the IoT and M2M market will have a crucial role in the future in addressing societal and economical needs, since they will reduce the gap between the physical and virtual worlds and, hence, create potential new business models based on new value opportunities, improve business productivity and security/safety of citizens. BETaaS platform allows an easy and controlled deployment and execution of IoT ready applications. But it can also benefit society, especially in the context of Smart Cities, with the creation of a myriad of applications and services on top data generated and collected by BETaaS platform. This provides the opportunity to create value-added services that can be realized as real-time analysis of context-rich data from multiple and heterogeneous data sources. This can be in the form of using public data sources, or things environments which are also able to actuate, that do not require the ownership of a physical infrastructure. Others models might be based in the use of private data sources and thing environments.

- its capacity to empower efficiency (seamless interaction of humans and machines seamlessly interacting and unintentionally collaborating as part of a greater system; with ubiquitous knowledge by being able to know everything (things, consumers, processes, etc.) in the environment of things/sensors/services. All made real by a Programmable World of Things.

There are a number of factors that have contributed to the fact that the IoT is starting to gain traction: cheaper sensors and more powerful devices; reduced communication and network costs while higher bandwidth and ubiquitous coverage options (e.g. 4G and WiFi networks); smartphones evolution and penetration; Big Data analytics as an enabler technology and the adoption of IPv6 to assure the creation of a network made of millions of devices. However, the real influence will be based on ramping up the network effect of connecting things and on developing apps.

Before there could be millions of cars there was the problem to build highway systems to avoid traffic jams and enable connectivity. We could identify the same problem for the IoT. First, there is the need of expanding network availability and connectivity in one hand, and in the other, the need of enabling the adoption of devices to the network. Then, the foreseen vision is that at the end, there would be a myriad of devices producing tons of data that can be used to make our lives easier and better; improve efficiency; and improve business and ordinary life processes, or completely creating new ones to fully leveraging the potential of the IoT. Therefore, in a few years from now it is expected to have millions of smart connected things onto our networks and the next step will be to make them become part of the choreography. They will be part of a coordinated part of a giant programmable machine put to the services for humans and business. Software will be a big part of the IoT market. This would be like a **Programmable World** in which applications will thrive to reach to the full potential of the IoT.

BETaaS is built to take the opportunity to provide an answer to some of the key trends and drivers found during the project ruling the economy and the expected influence of IoT. The expected evolution in the today's world is that the same global mobile and internet services leaders that now dominate the consumer battle are the best positioned to provide the platform-centric worlds that the IoT will likely need. Therefore the emergence of platforms and open standards (intentional or unintentional) will come into place, which will accelerate the pace of innovation. Therefore**, the emergence of tools and platforms that facilitate the creation of applications is expected to grow in the next future (software middleware or platforms that facilitate the connection, use, interaction of environments of heterogeneous things that master M2M data flow process (connecting devices/sensors; middleware for storing and securing the data, interacting and analyzing data; and finally providing interaction and feedback).** The emergence of multiple hardware proprietary platforms and technologies tightly coupled with specific mobile platforms will

difficult the emergence of innovative applications and hardware integration by smaller players unless open source platforms that are required to avoid proprietary siloed solutions are promoted and favored by the IT community. Until now, there are approaches undertaken by industry players, or acquisitions of independent companies that made big players to position in the market, opening the possibility for new comers to develop open platforms that allow multiple vendors and contexts. BETaaS does not only provide an horizontal approach to address this problem with **BETaaS runtime platform that simplifies the deployment and execution of content-centric M2M applications, with a horizontal abstraction. BETaaS also exploits the benefit provided by the use of Semantic technologies in the IoT context enables the emergence of high Level interaction and collaboration of things and platforms with BETaaS TaaS Reference Model**, which is expected to have impact to provide a high level integration with different platforms (mobile, devices and services) making very attractive the opportunity to build open source models over the integration of heterogeneous frameworks and platforms.

Another relevant aspect in which **BETaaS also provides value is in the future of hardware hubs for the IoT**. The need of gateways or hubs that connect multiple things and enable a closer layer of service is ramping up. There are various vendors providing specific appliances for different verticals, especially in the smart home (automation, energy and security). These specialized gateways are expected to increase the number of context for use and will embrace new things in the future. In this context, developers and vendors are demanding platforms that abstract them from the myriad of heterogeneous systems and different underlying technologies. BETaaS runs on multiple heterogeneous-hardware and provides the means to develop this kind of systems**. BETaaS also enables to provide value, expected in the age of the IoT, having the "cloud intelligence" living closer to the source of data and interactions capabilities of things.** This helps avoiding data latency; network optimization or unavailability; faster smart autonomous-response; and Big Data capabilities, using the concept of what is called "Fog Computing". It may also be possible to provide better security, privacy and control over sensitive devices and data within a localized environment.

According to the analysis of the market developed during the project, **BETaaS can benefit the following stakeholders in the ecosystem**, presented according to a supply vs. demand categorization of stakeholders of the ecosystem affected by both sides of the "platform model" provided by BETaaS. BETaaS identifies these two different market segments that have different requirements and needs and therefore need different approaches and positioning for BETaaS results.

- **Supply** (Industry, large organizations, SMEs and public administrations): is the segment providing hardware or things demanding possibilities to create value on top of their hardware: a) **Hardware Vendors (sensors, devices, boards, electronic devices and components, tools, other hardware, etc.)**: Hardware vendors; Hardware & Software Manufacturers; Telcos; Utility Companies (Things); Owners and Infrastructure providers of things environments (public or private);: and M2M Protocol Developer and Promoters (industry and standardization); and b) IoT Middleware: IoT System Integrators; IoT Platform Providers; IoT Service Providers; and Other SaaS, PaaS, BaaS providers

- **Demand**: is the segment eager of technology and tools to build and use IoT scenarios and create value around things and innovative services. It is encompassed mainly by developers, service providers, End-users and Consumers (Individuals and organizations); Software & Global Internet Services Companies; SDK and Development Tools and Services; Developers (Application & Services; solo developers and startups); and Application, Software and Service Companies (SMEs and large companies)

We also identify that **BETaaS could provide a relevant role in the ecosystem interacting with the next segments:** a) **Co-creation**: Distant market third parties companies (transportation, industry, commerce, logistics, energy, health, social, etc.); Verticals; and Competitor Alliances; and b) **Others** such as Standardization; IT industry community & Open Source Community; Research community; and Regulator & other government agencies.

BETaaS TaaS Reference Model and the Horizontal Adaptation Layer approach in conjunction are unique and none of the identified competitor solutions covers the full range of Abstraction, Semantic and Context-Awareness features provided by BETaaS Platform.

Moreover, the fact that BETaaS technologies will be shared as Open Source enables multiple options for value creation on top for IoT demand and supply segments competing alongside current market solutions.

4.3.2. Societal

Having BETaaS outcomes providing benefits to different stakeholders in the IoT and M2M ecosystem, puts the project in a good position to enable creating impact in society as well with the solutions and services that can be created using BETaaS technology. We now cite a few of foreseen potential opportunities identified.

BETaaS simplifies the way IoT solutions are built and operated impacting on tomorrow's society and people's everyday life. **Smart Cities** are needed in order to be able to comply with the demands of a growing number of citizens living in cities. People is more and more moving to cities; mega-cities and also small cities are numerous and rapidly growing in population. This future can only be sustained with "Sustainable Urbanization" as the key pillar, and this can only be achieved with an extensive use of technology, especially in the IoT and M2M context, making possible scaling and automation of existing services with the objective to grow and make it "physically and economically" possible to scale and serve citizens demands in future "Smart Cities". In this context, BETaaS enables the modernization and digitalization of public services (e.g. transport, utilizes, eHealth, etc.) and has already developed pilots in this context in conjunction with public administrations.

BETaaS also simplifies the **co-creation of value along the value chain, which empowers collaboration of services, creating a bigger system**, in order to take advantage of collaboration thanks to the possibilities that interoperability and abstraction of vertical heterogeneous systems and things environments working and interacting together.

Another relevant dimension in which BETaaS can provide value is in various contexts within what is called "**Smart Homes".** Besides home automation, improving energy efficiency in the home environment is one of the most important topics in which IoT is having more impact, grabbing more attention and growth. This has the objective **to try to improve our future as society in a "green" sustainable way.** And the IoT is driving radical change in the energy industry as consumers proactively adopt smart devices and smart energy solutions for the Smart Home. The growing penetration of this kind of devices and home appliances connecting to several other things within the home environment is changing the way that utilities execute energy efficiency and demand response programs, which is thriving co-creation models between users and providers, thanks to the combination of data and smarter products enable various new approaches to energy management for the **Smart Home** context.

Another aspect that **IoT is clearly increasing its impact in everyday lives of healthcare consumers and providers**. IoT enables providing services for rapid treatment, preventive and self-care services; and also for community-based health and social services. The market is currently growing with the emergence of a myriad of sensor-based wearable technologies and the capacity to analyze the data and provide smart feedback, and in this context BETaaS provides the benefit to allow multiple hardware vendors to collaborate and be used by existing or new services.

4.4. Main Dissemination activities and exploitation achievements

BETaaS partners worked towards full awareness in Europe by participating and disseminating the work of the project and its results using diverse international, national and regional channels - in conferences, meetings and workshops. In addition, project website, brochures and posters were prepared in English and the languages of the end-user partners for promotion of the project. Project partners also established strong and fruitful links with other relevant European projects, in order to exchange views and experiences with them. Last but not least, substantial focused dissemination efforts targeted national and regional stakeholders. Major dissemination activities are listed in table A1 and A2 of [10] .

In this last reporting period M13-M30, taking into account the recommendations during the Second Project Review, in this last reporting period BETaaS consortium has increased all WP7 related activities with the objective to improve and create as much impact as possible.

In terms of Exploitation, the consortium has increased its activity, both at individual and joint level. It has also taken the lead in some dissemination activities that impact in the task of exploitation and sustainability with the project results, especially in Dissemination for visibility, outreach and stakeholder involvement and community building in order to make BETaaS take a more ambitious approach and achieve the expected impact.

In terms of Standardization, the consortium has used it as yet another channel for dissemination, to share the BETaaS experience in different initiatives, while exploitation have been improved (as well as dissemination ones) in order to make BETaaS ambitious to achieve the expected impact.

5. Partners and contacts

5.1. List of beneficiaries

No	Name	Short name	Country
1	INTECS SPA	INTECS	Italy
2	ATOS SPAIN SA	ATOS	Spain
3	CONVERGE ICT SOLUTIONS AND SERVICES AE	CONV	Greece
4	AALBORG UNIVERSITET	CTIF	Denmark
5	HEWLETT PACKARD ITALIANA SRL	HP	Italy
6	FUNDACION TECNALIA	TECN	Spain

	RESEARCH & INNOVATION		
7	UNIVERSITA DI PISA	UPI	Italy
8	PARTECIPAZIONI TECNOLOGICHE SPA	INCS	Italy

5.2. Coordinator contact details

Ms. Novella Buonaccorsi
Program Manager for
INTECS S.p.A
Via U. Forti, 5A
56126 PISA
Tel: *+39.050.96.57.411*
Fax: *+39.050.96.57.400*
E-mail: *novella.buonaccorsi@intecs.it*

5.3. Project logo

5.4. Project website

The project website is accessible through: http://www.betaas.eu/


www.betaas.eu
BETaaS
Home
Our Project
Consortium
Deliverables
News
Publications
Webinars
Collaboration
Contacts
Building the Environment for the Things as a Service
BETaaS [bē'tas] is an FP7 STREP project co-funded by the European Commission under the ICT theme (Call 8) of DG CONNECT.
Consortium
This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 317674
© 2014 BETaaS Consortium | template inspired from styleshout
Home | XHTML | CSS

6. References

[1] E. Mingozzi, G. Tanganelli, C. Vallati, V. Di Gregorio, "An open framework for accessing Things as a service," *Proceedings of the 16th International Symposium on Wireless Personal Multimedia Communications* (WPMC 2013), Atlantic City, NJ, USA, June 24-27, 2013.

[2] Deliverable D1.4.2. TaaS Reference Model.

[3] Deliverable D2.1.3. Basic Capabilities and Content Use.

[4] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, September 2005.

[5] E. Mingozzi, G. Tanganelli, C. Vallati, "A framework for Quality of Service support in Things-as-a-Service oriented architectures," Journal of Communication, Navigation, Sensing and Services (CONASENSE), Vol. 1, No. 2, pp. 105-128, August 2014.

[6] Deliverable D2.2.3. Specification of the extended capabilities of the platform.

[7] G. Tanganelli, C. Vallati, E. Mingozzi, "Energy-Efficient QoS-aware Service Allocation for the Cloud of Things, Proceedings of the IEEE Workshop on Emerging Issues in Cloud (EIC 2014)" - co-located with IEEE CloudCom 2014, Singapore, December 15-18, 2014.

[8] An Efficient Protocol for Authenticated Key Agreement. Law, Laurie and Menezes, Alfred and Qu, Minghua and Solinas, Jerry and Vanstone, Scott. 2, Norwell, MA, USA : Kluwer Academic Publishers, March 2003, Design, Codes, and Cryptography, Vol. 28, p. 16.

[9] B. Anggorojati, N. R. Prasad, R. Prasad, " Secure Capability-based Access Control in the M2M Local Cloud Platform", 3rd International Conference on Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE) 2014, Aalborg, Denmark, May 11-14, 2014.

[10] Deliverable D8.3 BETaaS - Final Report