EAI Endorsed Transactions
on Cloud Systems
Research Article ICST.ORG

BETaaS platform – a Things as a Service Environment for future M2M marketplaces

Bayu Anggorojati[1,*], Sofoklis Kyriazakos[1], Neeli Prasad[1], Carlo Vallati[2], Enzo Mingozzi[2], Giacomo Tanganelli[2], Novella Buonaccorsi[3], Nicola Valdambrini[3], Belén Martínez[4], Francisco Nieto De-Santos[5], Nikolaos Zonidis[6], George Labropoulous[6], Alessandro Mamelli[7] and Davide Sommacampagna[7]

[1] Center for TeleInfrastructure, Aalborg University, Denmark
[2] Department of Information Engineering, University of Pisa, Italy
[3] Intecs S.p.A., Italy
[4] Tecnalia Research & Innovation, Spain
[5] ATOS Research & Innovation, Spain
[6] Converge ICT Solutions & Services S.A., Greece
[7] Hewlett-Packard Italiana S.r.l., Italy

Abstract

Building the Environment for Things as a Service (BETaaS) is a novel platform for the deployment and execution of content-centric Machine-to-Machine (M2M) applications, which relies on a local cloud of gateways. BETaaS platform provides a uniform interface and services to map content with things in a context-aware manner. 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, BETaaS platform is based on a suitable defined Internet of Things (IoT) model, allowing the integration of the BETaaS components within the future Internet environment. In this paper we present the BETaaS concept, the high level platform architecture and application scenarios that extend the state-of-the-art in M2M communications and open the horizon for future M2M marketplaces.

Keywords: IoT, M2M, Local cloud, TaaS.
Received on 16 December 2014, accepted on 20 January 2015 , published on 23 February 2015

doi: 10.4108/cs.1.1e2

1. Introduction

Marketplaces of applications became a trend over the past years within this decade and this trend is a game-changer by providing a different business model. The app marketplaces are gaining more popularity thanks to the massive developments and usage of mobile apps. But mobile apps world is not the only place where app marketplaces exist. Recently, computer's operating systems make use of such models, for instance Mac App Store[†] and Ubuntu Software Center[‡]. Even TV manufactures use this 'channel' to differentiate from the competitors in the smart TVs world.

M2M communication which implies the elimination of the human in the loop is not only a trend, but is the natural evolution of application marketplaces. The IoT that integrates objects into the Internet has opened new horizons in the M2M communications and we consider this as a unique opportunity to come up with a Things-as-a-Service environment that will increase the spectrum of M2M applications.

In order to make such idea to happen in the M2M world, several critical requirements need to be put in place. First of all, there needs to be an open and generic set of Application Programming Interfaces (APIs) to expose the services provided by the smart objects or *things* connected in the platform to the applications. However, creating such common programming interface in a M2M platform is not a trivial thing considering the heterogeneity of the M2M system. For the sake of simplicity, the term smart object that refers to the M2M physical devices, such as sensors and actuators, will simply be referred as *things* throughout this paper.

It leads to a necessity of unifying the way *things* are represented within the platform. Moreover, the vertical and closed approach that is currently adopted in the

[†] http://www.apple.com/osx/apps/app-store/
[‡] https://apps.ubuntu.com/cat/applications/software-center/

*Corresponding author. Email:ba@es.aau.dk

M2M system implementation, limits the types of application that can be delivered to the end users. Apart from the aforementioned technical requirements and issues, other requirements, such as security and privacy, big data, context awareness and Quality of Service (QoS), need to be provided as well in order to create a complete M2M platform that enables M2M marketplace.

BETaaS aims at overcoming such limitations through creating a horizontal runtime platform of M2M system. Although several horizontal M2M platforms have been recently designed and developed, BETaaS platform distinct itself from the other horizontal M2M platforms in its architecture that is based on a distributed runtime environment consists of a so-called *local cloud* of *gateways*, allowing access to *things* connected to the platform regardless of their technologies and physical location. The benefit of such architecture is that not only it enables the deployment of private and isolated platforms, but also allows applications to run close to the physical M2M *things* deployment, which is a critical factor for M2M applications that require short response time and fresh information from the *things*. This closeness also helps BETaaS platform to efficiently manage the context of things, being this context a key element for the creation of applications.

In this paper, our technical approach in BETaaS to enable M2M marketplaces will be further discussed. In section 2, the requirements of M2M applications and IoT as well as the approaches currently used by the existing platforms will be presented, which will lead to gap analysis between the requirements and existing approaches. In section 3, the approach as well as the technical detail, particularly the high level architecture and the Things as a Service model, of BETaaS platform will be elaborated. In section 4 we focus on M2M applications that can be available in future marketplaces, requiring the flexibility offered by BETaaS approach. Finally, in the conclusions section we focus on the next step and the vision of BETaaS in the IoT and M2M world.

2. M2M applications and marketplace

2.1. Technical requirements

First and foremost, an M2M marketplace needs an open platform that allows both the things to be easily connected to the platform and the application to be developed on top of the platform. Connecting *things* into a single platform is a big challenge due to its heterogeneous nature. There are literally wide ranges of *things'* types, capabilities, and access technologies; from the sensors and actuators powered with (or without) battery which are using Bluetooth, ZigBee, ZWave, and others as their access technologies; to Radio Frequency Identification (RFID) that requires no battery power. The standards and communication protocols also varies from the ETSI M2M [2], Constrained Application Protocol (CoAP) [3], and Message Queue Telemetry Transport (MQTT) [4], just to name a few, and even some proprietary standards. To this end, a layer that provides common interfaces to the higher layer component in the platform regardless of the underlying M2M technologies at the physical level is certainly required. It is definitely one of the critical point in order to reach broader scope of M2M applications using whatever types of M2M devices.

On top of that, a contextual management of the information that characterizes the connected *things* should also be part of the platform. For instance, the platform should be able to identify the type of thing (e.g. sensor or actuator), the type of provided information (e.g. temperature, humidity, presence, GPS coordinate, etc), its location, its battery level, and other necessary information coming from the heterogeneous M2M system at the lower layer. It thus allows an application to request for information according to its requirements no matter who (i.e. any particular *things*) provides them.

As earlier mentioned, the platform needs to provide a set of common interfaces for external application to access the platform. In this regard, the interface needs to be application developer "friendly". For instance, it needs to be based on RESTful interface, as this is the most popular method used by the web and mobile developers – which are the most potential markets for the M2M applications. On the other hand, the platform should also provide other types of interfaces to give more choice to the application developers.

Apart from the specific requirements of M2M marketplace, other technical requirements from the point of view of the M2M applications need to be fulfilled too. In addition to heterogeneity, IoT and M2M applications are also characterized by mobility, high number of nodes, and some may require low latency. To this end, a platform with distributed architecture as the opposite of the centralized cloud computing approach that is adopted by most – if not all – of the prevalent M2M platforms nowadays.

2.2. Current approaches

The most common approach in developing M2M application is the *vertical* isolated approach, meaning that the application is developed upon particular

software and hardware that allows no cooperation with each other to share *things* capabilities for efficiency. Another common practice in M2M application development is that each infrastructure serves a single M2M application, excluding any interoperability pattern between other systems, applications or *things*. Thus, when new M2M applications are needed, providers have to re-create M2M communication platforms and data representation formats. Typically it is a custom made application for a specific M2M application, developed by smart object manufacturers or system integrators. Certainly, this approach hinders the participation of software developers that have little to no knowledge on the specific M2M technologies, and thus hinders the penetration and widespread of M2M system usage.

To break out from such common M2M application development pattern, several horizontal M2M platforms have recently been developed. For instance, *OpenIoT*[§] project delivered an open source middleware based on cloud environments for IoT and offering utility-based (i.e. pay-as-you-go) IoT services. *COMPOSE*[**] project aims at integrating, publishing and sharing data from Internet-connected objects (i.e. IoT) into services and applications to create an open marketplace. A number of horizontal cloud-based M2M platform are also commercially available, for example *Xively*[††], *Libelium*[‡‡] and *ThingWorx*[§§]. In 2013, ThingWorx launched the first marketplace for the Internet of Things that enables developers, hardware and software providers, and system integrators to build value-added IoT/M2M components and make them available to broad range of companies [5].

Even though similar vision of creating M2M marketplace built on top of horizontal M2M platform has recently existed, they are based on centralized cloud architecture, which is difficult to cope with the M2M characteristics such as high mobility, high number of nodes, and low latency in some of M2M applications. In response to this challenge, a novel approach towards a geographically distributed architecture has been defined recently. This novel architecture referred as *Fog Computing* [6], provides the computing, storage, and network services between end devices and centralized cloud computing data centers, typically, but not exclusively at the edge of the network in order to efficiently support M2M applications. Although decentralized approach is acknowledged as the long-term evolution to support M2M applications [7], only a few solutions have been proposed in literature. However, they are usually bounded on a specific technology, e.g. exploit the CoAP protocol [8], or provide only the basic set of functionalities to applications, e.g. focus only on interoperability and integration [9].

3. BETaaS platform

3.1. Concept and architecture

BETaaS is providing a new vision about the way to expose and manage things in the Internet of Everything environment through distributed runtime architecture, i.e. local cloud of gateways, and exploiting semantic technologies to support content-centric M2M applications execution and context awareness for heterogeneous M2M systems. BETaaS platform seamlessly integrates existing heterogeneous M2M systems composed of different *things* by means of adaptors within a gateway. Each gateway runs the BETaaS run-time environment that forms a logical overlay. The logical federation of networks forms a *local cloud* in which each gateway shares the functionalities offered by the *things* of its M2M systems with the rest of the network. The term *local cloud* referring to the set of gateways hosting the platform has been adopted to highlight the locality of such deployments, often physically confined in space, and because of some inherent characteristics of the general cloud systems that are incorporated in the BETaaS platform, such as *resource pooling* (i.e. physical things providing required services based on context information and are transparent to the application), *rapid elasticity* (i.e. high scalability and ability to handle burst request thanks to distributed architecture), and *measured service* (i.e. service execution based on the current status of physical devices hosting the gateways). On top of that, the run-time platform running on each gateway provides M2M applications connected to any of the BETaaS gateways a common interface to access their respective *things*, irrespective of location and underlying M2M technology. At this point, the external application sees BETaaS platform as a single instance in a form of local cloud, i.e. seamless integration of heterogeneous M2M system as well as the network of gateways. To clearly understand the BETaaS concept, please refer to the right hand side of **Figure 1**.

The BETaaS high level architecture (see the left hand side of **Figure 1** and [1] for more details) reflects

§ http://openiot.eu
** http://compose-project.eu
†† http://xively.com
‡‡ http://www.libelium.com
§§ http://thingworx.com

the previously mentioned concept by means of equivalent layered structure. Such structure guarantees the proper level of abstraction to applications at the top and the flexibility necessary to integrate different system characterized by different technologies at the bottom. At the bottom level, BETaaS integrates heterogeneous M2M systems at the Physical layer into a unified M2M system thanks to the *Adaptation layer*. The Adaptation layer allows plugins to be dynamically added and deployed into the platform to accommodate different M2M systems, e.g. ETSI M2M, CoAP, and even proprietary systems, within the platform. The plugins provide interfaces to the higher layer components and convert the request from the later into their respective M2M systems. On top of the *Adaptation layer*, *Things-as-a-Service* (*TaaS*) *layer* is there to provide an abstract and uniform description of the underlying M2M systems regardless of the technology, communication protocol, and physical location. The TaaS paradigm defines for each thing connected to the platform, a *thing service* is created to represent the *basic service*(s) that can be exposed to applications in a content-centric manner. In order to allow the application to transparently access *thing services* irrespective of the gateway providing physical *thing* connection, TaaS layer is implemented in distributed fashion to cooperatively share resources among gateways, thus achieving the local cloud concept as earlier mentioned.

On top of *TaaS layer*, *Service layer* is put forward in the platform, defining the platform interface to the external applications. In general, *Service layer* exposes all of the available *thing services* in the platform as *basic services* allowing applications to interact with the *things* connected to the platform. In addition to basic services, this layer also allows dynamic deployment of custom services, called *extended services*, from a third party. Extended service can be used to extend or combine the functionalities of the basic services provided by the platform through implementation of complex logic tailored to a specific running instance of BETaaS platform. Not only the support of extended service deployment in the platform allows wide range of applications endorsement through customization in the platform, but also opens the opportunity to establish a *marketplace* of extended services that can be installed on demand by the end users.

In order to illustrate the distributed nature and flexibility of the platform, the left hand side part of **Figure 1** shows a number of gateways forming a local cloud. Further, it shows gateways with different BETaaS components (and not) already equipped and how to make them BETaaS enabled. The first three gateways, i.e. GW1 – GW3, are called BETaaS-Aware because they have at least one BETaaS layer implemented. GW1 has all the BETaaS layers thus it is BETaaS enabled by default, while GW2 and GW3

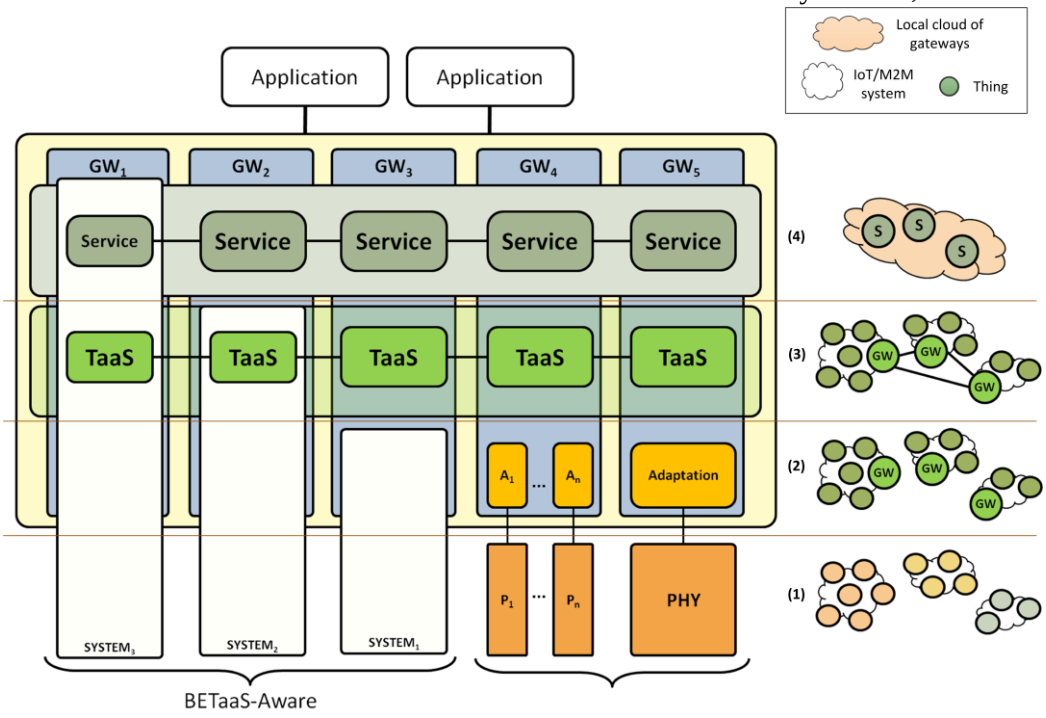

Figure 1. BETaaS concept and high level architecture

only implement TaaS+Adaptation layers and Adaptation layer respectively which require them to install the missing BETaaS layers components to make them BETaaS enabled. On the other hand, GW4 and GW5 are both BETaaS-Unaware, thus installing all the BETaaS layers components are needed. Here we can see an example of two gateways with different Adaptation layer components. GW5 only has one kind of Adaptation layer component meaning that it can only accommodate a corresponding M2M system, while GW4 can support different M2M systems because it is equipped with several corresponding Adaptation layer components.

The advantage of such approach over the centralize cloud approach from the perspective of system deployment and marketplace is that any people, community, or organization can have their own BETaaS local cloud (with the option to connect to centralize cloud). It is thus increasing the range of application types to be developed and promoted in the marketplace. Furthermore, it also allows stakeholders to join the ecosystem; not only the external application developers but also device vendors, things developers, and extended service developers.

3.2. Platform features

Additional to the novel concept and architecture, BETaaS platform is equipped with advance features embedded by design. Such features will be described in the following sub-sections.

3.2.1. Context-aware support

The BETaaS platform is content-centric in the sense that it provides services that depend on the type of data that they provide and on the context in which that data is used. The circumstances that are considered in BETaaS as part of the context of a thing are its battery level, its available computing capacity, the communication protocol used and its location. BETaaS uses semantic technologies and natural language processing to unify the information that comes from heterogeneous things, to infer new knowledge from raw data in a context-aware fashion, and to generate unique *thing services* for each of the things connected to the platform.. Following the purpose of unifying information, an ontology has been built: the BETaaS ontology. Thishe 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: SSN[***], OWL-Time[2][†††], CF[3][‡‡‡], Phenonet[§§§], MUO[****], FIPA[††††] and GeoNames[‡‡‡‡]. When a new thing is connected to the BETaaS platform, all the information related to that thing (e.g. type of thing, contextual data, etc.) is inserted in the BETaaS ontology. To promote standardization in the IoT field, the BETaaS ontology is populated by means of common vocabularies whenever possible. More precisely, we have taken advantage of the significant efforts in standardization and interoperability made by WordNet,[§§§§]. a lexical database that groups English words into sets of synonyms (synsets). Whenever possible, the information related to things is translated to WordNet sysnsets before storing it in the ontology. 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[*****], which offers a common data model to organize classifications in a hierarchical way. The relationships between the 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 things 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). The contextual information associated to each of the things connected to a gateway allows the platform to create a *thing service* for each of those things, following the nomenclature *setLocationType*/*getLocationType* (e.g. *getKitchenTemperature*).

3.2.2. QoS management

Support for heterogeneous QoS requirements is a non-trivial challenge, considering the broad variety of applications that can run on the BETaaS platform. Classic approaches define a standard QoS model to categorize QoS requirements into a pre-defined set of service classes, e.g. [10]. Since at run-time applications can only select one class with a fixed set

[***] http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628
[†††] http://www.w3.org/2006/time
[‡‡‡] http://www.w3.org/2005/Incubator/ssn/ssnx/cf/cf-property
[§§§] http://www.w3.org/2005/Incubator/ssn/ssnx/meteo/phenonet
[****] http://purl.oclc.org/NET/muo/muo
[††††] http://www.fipa.org/specs/fipa00091/PC00091A.html
[‡‡‡‡] http://www.geonames.org/ontology/ontology_v3.1.rdf
[§§§§] http://wordnetweb.princeton.edu
[*****] http://www.w3.org/2004/02/skos/intro

of service parameters, supporting a wide range of applications will increase dramatically the complexity.

In order to reduce the platform complexity, a simple schema composed by three service classes has been adopted: Real-time service (applications with hard response time requirements), Assured service (applications with soft response time requirements) and Best-effort service (applications that do not require any assurance). At the same time, flexibility is guaranteed allowing applications to customize their requirements through a dynamic negotiation procedure within the selected service class. The negotiation is performed at the time of the installation following a two-stages procedure, as described in [11]: first the application specifies the QoS parameters required for the service, then the service negotiates with the TaaS layer the QoS of the thing services required to fulfill application requirements. For the sake of interoperability, a standard Service Level Negotiation (SLA) interface that allows complex developments is adopted, the WS-Agreement Negotiation protocol, which is the de-facto standard for SLA agreement negotiating, establishing and managing for Web Services.

Once the negotiation phase is performed a SLA is established and applications could invoke thing services with the negotiated QoS level. Specific requirement of the proposed platform is the exploitation of the possibilities offered by equivalent things, things that can provide interchangeably the same Thing Service according to context information. As result of the integration of different systems, large IoT networks are expected to be 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 in its design can take full advantage of this variety is included in the platform to allow efficient management of resources. Such QoS framework is implemented in the platform through a two-phase procedure, namely, *reservation* and *allocation*. The reservation phase performs admission control and, most importantly, manages resource reservation by exploiting equivalent thing services. The allocation phase, instead, performs allocation of resources at time of thing service invocation. The latter is implemented to optimize the allocation by means of a number of parameters, e.g., in terms of energy efficiency. For a more detailed description of overall QoS framework included in BETaaS, we remind the interested reader to [12].

3.2.3. Big data management

Data management is an important feature of BETaaS platform due to the fact that a lot of data, both in volume and frequency, is generated by things and consumed by the applications. In order to manage large amount of data, BETaaS platform provides capabilities to manage data in distributed manner – exploiting distributed architecture of the platform – and across different layers. Such a strategy is considered because a running BETaaS instance might be composed of gateways with constrained and unconstrained devices, i.e. different computational power and storage capability, a gateway hosted in a constrained device may have only data management capability at the TaaS layer which is responsible to collect data from things, perform simple adaptation of data structure, and deliver it to gateway with more advance data management capability or a dedicated storage. On the other hand, more capable gateway can have data management at the service layer which can perform more advance tasks, such as scheduled processing of data or real time query, i.e. when resource is available, and provide them to the applications. Additionally, some gateways can have more storage capabilities where they can receive and store data from more constrained gateways, and further process large amount of data, thus enabling big data management. Such gateways can also perform data analytic functionalities upon large amount of collected data in a form of real time query or batch processing. The storage of data is based on a distributed file system, so that replication, parallelism, and high availability are guaranteed. The big data feature of the platform is exposed to applications through a specific module of big data manager that provides an interface, named *data task*, which allows analytics deployment and other tasks that were previously described.

3.2.4. Security management

Security management in BETaaS mainly deals with the access control to the sensible data and trust evaluation of things and gateways. A capability-based approach for access control that includes access delegation feature is used. The approach is coupled by a Public Key Infrastructure (PKI), which is practically implemented through digital certificates. With this approach, an application developer will receive a certificate signed by BETaaS' trusted Certificate Authority (CA) upon requesting to use BETaaS APIs through a registration process.

During installation, as the application has obtained the certificate, it acquires a capability or token which states the access rights to the thing services, such as access conditions, validity period, delegation information, and digital signature. The platform evaluates the access policies according to the required thing services by application , which results in grating a set of tokens to application. Every time a service is invoked, the token is verified by validating the digital signature, validity condition, delegation chain, and relevant access rights and conditions. Complete

mechanism of BETaaS access control mechanism can be reviewed in [13]

Relying on external systems manufactured and maintained by independent third-parties can raise trust issues. For this reason the BETaaS platform includes a trust model to monitor things and gateways behavior evaluating their reliability. The trust model takes into account: the security mechanisms available for interacting with entities, the QoS fulfillment, dependability measures related to things and gateways, scalability as interactions increase, expected availability because of battery load, stability in data generation and gateways reputation.

3.2.5. Virtualization management

Virtualization feature is included in the BETaaS platform for two main purposes: to provide a way for deploying applications locally (in an isolated environment, protecting the core BETaaS platform) and to enable scalability for the platform functionalities (such as computation and storage for big data analysis).

The platform exploits both local virtualization capabilities provided by gateways and external cloud resources provided by third parties. This is achieved by providing a set of basic images that contain pre-installed software depending on their purpose. BETaaS provides an image for big data computation and storage nodes, as well as an image with a Java web container for web applications deployment. There is also an image for virtualizing gateways.

3.2.6. Extended service support

As explained earlier, BETaaS allows deployment of extended service as an extension of the basic services provided in a specific running instance of the platform. For example, extended service can leverage basic thing services in a BETaaS instance deployed in a smart home environment which consists of domotic system, home security, smart meter or smart grid, and home entertainment. It is implemented as a software bundle that can be dynamically installed, run, updated, disabled, and uninstalled at run time. At the operation level, extended services may operate as automatic process or can expose a set of new interfaces, i.e. in addition to the basic thing services, that can be exposed to external applications. The latter case increases the possibility of more advance applications to be developed even by software developers that have limited knowledge about the platform itself, thus exploiting the usage of M2M marketplace even more.

4. Application scenarios

In order to show what types of applications can be developed on top of BETaaS platform, two application scenarios, namely smart city and smart home scenarios, will be presented in this section.

4.1. Smart city

The Smart City scenario is mainly focused on the integration of Smart Things belonging to different systems. This scenario shows how BETaaS can adapt to different data sources, how it can be used to build a complete system and how it can provide an added value to existing ones.

4.1.1. Building blocks

The scenario is built upon:

- A smart lighting system that controls public lamp posts in a parking lot
- A system to manage cars access to a restricted area of the city based on the car's positions
- A traffic system that receives data from traffic sensors installed along a network of roads.

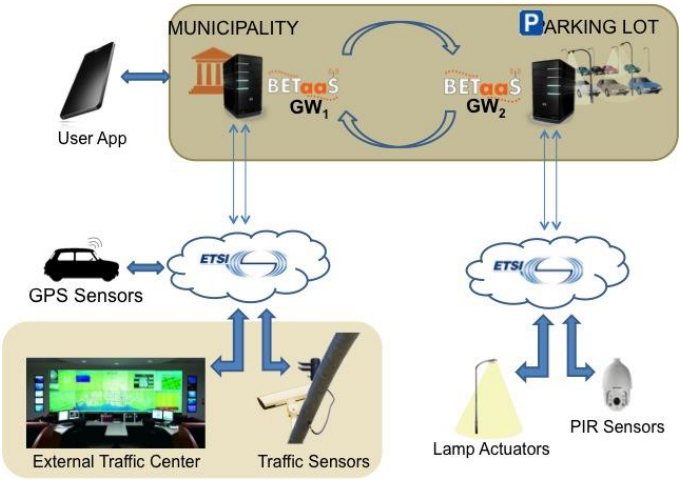

Figure 2. Smart city scenario deployment on BETaaS platform

Figure 2 illustrates the building blocks of the smart city scenario deployment on BETaaS platform. It shows two different systems owned by different organizations or entities and each of them has a BETaaS GW connected with a set of smart things through ETSI M2M standard. The second BETaaS gateway at the parking lot implements smart lighting system through a single BETaaS gateway connected to ETSI-enabled devices: light intensity tuneable lamp posts and infrared presence sensors. The lamp post tunes its light intensity automatically depending upon the presence of people at the parking lot (based on the input from Passive Infra-Red (PIR) or presence sensor)

and the traffic data from different system. The first BETaaS gateway, owned by the municipality, is in charge of receiving users' position through a GPS receiver and matching them to the city map. It is intended to implement a service to check the access to Low Emission Zones (LEZ), making users to pay a fee as they enter. The traffic system is a completely independent system called SIMIS (Sustainable and Intelligent Mobility Integrated System) [14]. It allows checking the current status of roads and parking lots, also providing traffic data to external applications, i.e. the smart lighting system at the parking lot. In this scenario it is shown how BETaaS can be easily extended to receive data from it.

4.1.2. Integration through BETaaS

In this smart city scenario the two BETaaS gateways described above are then used to make up a single BETaaS instance. Once they are configured to join each other, they start sharing their resources. Gateways resources are mainly represented by the Thing Services they created on top of the Things: lamp posts, presence sensors, traffic sensors and cars' GPS receivers. BETaaS allows adding application logic inside the instance through extended services. In this scenario two extended services are included:

- The smart lighting extended service on the first gateway. It not only exploits the presence sensors directly connected to the gateway itself but it also considers the traffic data provided by the other gateway. Lamp light is then intensified or dimmed also based on the current traffic density.
- The LEZ extended service not only uses the cars' position to make users pay a fee once they enter the restricted zone. It also exploits traffic data coming from SIMIS to compute dynamic fees based on the current roads congestion.

Extended services may also be accessed by BETaaS users from their external applications. So in this case users access the platform through mobile applications from their cars, being notified about the current fee that is currently applied, depending on their position and traffic intensity. **Figure 3** presents a screen shot of LEZ BETaaS mobile apps which shows different rates applied to the user/car traveling within the LEZ area in different traffic intensity, i.e. high, medium, and low.

Figure 3. Screen shot of LEZ BETaaS mobile apps

4.2. Smart home

The smart home scenario is focusing on a typical need of Smart Homes and Building Management Systems, which is the exploitation of existing infrastructures. BETaaS concept aims to prove through this scenario, that it can extend proprietary systems in order to result in integrated solutions with multiple services and capabilities. The deployment of smart home scenario application on BETaaS platform is shown in **Figure 4**.

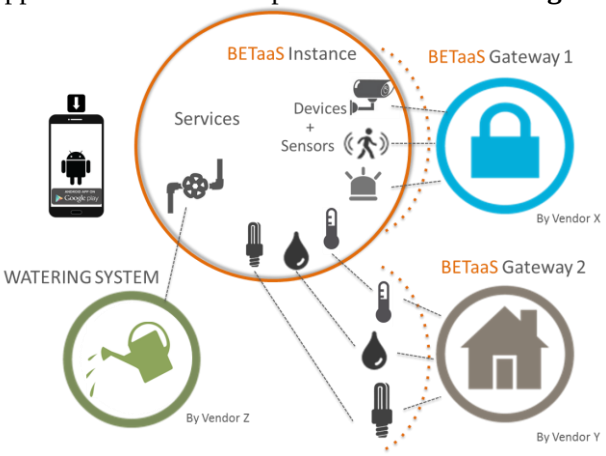

Figure 4. Smart home scenario deployment on BETaaS platform

In this scenario, the smart home consists of two systems, namely home security system provided by Vendor X and domotic system provided by Vendor Y. Both systems are originally proprietary systems, but they are made to be BETaaS enabled by installing BETaaS components in their gateways. Once both systems are BETaaS enabled, they can create a

BETaaS instance (i.e. local cloud) and share their resources (things), e.g. CCTV, presence sensor, and building access control from BETaaS gateway 1, and temperature sensor, humidity sensor, and lamp from BETaaS gateway 2, to enable richer sets of services exposed to external applications. Then, the user needs a watering system for his garden, so he bought an additional valve and download “Watering App” from BETaaS marketplace. The Watering App takes the information from the thing services exposed by the existing BETaaS instance in his home. Further, it enables automatic garden watering based on the time of the day, the temperature and humidity provided by gateway 2 and presence of person at home provided by gateway 1.

5. Conclusion

The paper presents the Things as a Service environment for future M2M marketplaces, proposed by BETaaS. The technical requirements and approach of BETaaS are described, as well as the indicative scenarios and applications. Unlike many IoT and M2M approaches, BETaaS is a tangible platform that aims become an open-source community to further develop the platform, provide good documentations, transfer the knowledge as well as technical support to the community, and achieve the highest impact. We strongly believe the clear advances of BETaaS platform and some of the future works we consider, such as establishing the community and partnerships with all stakeholders, will create high influence in the evolution of M2M applications and marketplace.

Acknowledgements.

This work has been carried out within the activities of the project "Building the Environment for the Things as a Service (BETaaS)", which is co-funded by the European Commission under the Seventh Framework Programme (grant no. 317674).

References

[1] Mingozzi, E.; Tanganelli, G.; Vallati, C.; Di Gregorio, V., "An open framework for accessing Things as a service," Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on , vol., no., pp.1,5, 24-27 June 2013.

[2] ETSI TS 102 921, "Machine to Machine Communications (M2M); mIa, dIa and mId interfaces," European Telecommunications Standards Institute (ETSI), 2012.

[3] Z. Shelby, K. Hartke and C. Bormann, "The Constrained Application Protocol (CoAP)," Internet Engineering Task Force (IETF), 2014.

[4] A. Banks and R. Gupta, Eds., MQTT Version 3.1.1, OASIS Standard, 2014.

[5] S. Schwind, "ThingWorx Launches the First Marketplace for the Internet of Things," 18 November 2013. [Online]. Available: http://www.reuters.com/article/2013/11/18/pa-thingworx-idUSnBw185503a+100+BSW20131118.

[6] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog computing and its role in the internet of things," in *MCC '12 Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, New York, 2012.

[7] S. Abdelwahab, B. Hamdaoui, M. Guizani and A. Rayes, "Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler," *Internet of Things Journal, IEEE*, vol. 1, no. 3, pp. 276 - 288, June 2014.

[8] L. Mainetti, V. Mighali, L. Patrono and P. Rametta, "Discovery and Mash-up of physical resources through a Web of Things architecture," *Journal of Communications Software and Systems*, vol. 10, no. 2, 2014.

[9] C. Sarkar , A. U. Nambi , R. V. Prasad and A. Rahim, "A scalable distributed architecture towards unifying IoT applications," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, Seoul, 2014.

[10] Mef, Marie-Aurélie, et al. "Enabling QoS in the Internet of Things." CTRQ 2012, The Fifth International Conference on Communication Theory, Reliability, and Quality of Service..

[11] Mingozzi, E.; Tanganelli, G.; Vallati, C., "A framework for QoS negotiation in things-as-a-service oriented architectures," Wireless Communications, Vehicular Technology, Information Theory and Aerospace & Electronic Systems (VITAE), 2014 4th Internationa.

[12] 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, May 2014..

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

[14] Intecs SpA, "SIMIS (Sustainable and Intelligent Mobility Integrated System)," [Online]. Available: http://www.intecs.it/pdf/SIMIS.pdf.