Noname manuscript No.
(will be inserted by the editor)

BETaaS: a platform for development and execution of Machine-to-Machine applications in the Internet of Things

Carlo Vallati · Enzo Mingozzi · Giacomo Tanganelli · Novella Buonaccorsi · Nicola Valdambrini · Nikolaos Zonidis · Belen Martinez Rodriguez · Alessandro Mamelli · Davide Sommacampagna · Bayu Anggorojati · Sofoklis Kyriazakos · Neeli Prasad · Javi Nieto De-Santos · Oliver Barreto Rodriguez

Received: date / Accepted: date

Abstract The integration of everyday objects into the Internet represents the foundation of the forthcoming Internet of Things (IoT). Smart objects will be the building blocks of the next generation of applications that will exploit interaction between machines to implement enhanced services with minimum or no human intervention in the loop. A crucial factor to enable Machine-to-Machine (M2M) applications is a horizontal service infrastructure that seamlessly in-

Carlo Vallati, Enzo Mingozzi, Giacomo Tanganelli
Department of Information Engineering, University of Pisa Via Diotisalvi, 2. I-56122 Pisa. Italy. E-mail: {c.vallati, e.mingozzi, g.tanganelli}@iet.unipi.it

Novella Buonaccorsi, Nicola Valdambrini
Intecs S.p.A. Via U.Forti 5, 1-56121, Pisa. Italy. E-mail: {novella.buonaccorsi, nicola.valdambrini}@intecs.it

Nikolaos Zonidis
CONVERGE ICT Solutions & Services SA 74 Panormou Street, 125 23 Athens, Greece. E-mail: nzonidis@converge.gr

Belen Martinez Rodriguez
Tecnalia Research & Innovation Parque Tecnolgico de Bizkaia, E-48170 Zamudio. Spain. E-mail: belen.martinez@tecnalia.com

Alessandro Mamelli, Davide Sommacampagna
Hewlett-Packard Italiana s.r.l. Via G. Di Vittorio 9, 20063 Cernusco sul Naviglio (MI), Italy. E-mail: {alessandro.mamelli, davide.sommacampagna}@hp.com

Bayu Anggorojati, Sofoklis Kyriazakos, Neeli Prasad
Center for TeleInFrastruktur, Aalborg University Fredrik Bajers Vej 7, DK-9220 Aalborg st. Denmark. E-mail: {ba, sk, np}@es.aau.dk

Javi Nieto De-Santos, Oliver Barreto Rodriguez
ATOS Research & Innovation C/ Albarracn, 25, 28037 Madrid, Espaa. E-mail: {francisco.nieto, oliver.barreto}@atos.net

tegrates existing IoT heterogeneous systems. The authors present BETaaS, a framework that enables horizontal M2M deployments. BETaaS is based on a distributed service infrastructure built on top of an overlay network of gateways that allows seamless integration of existing IoT systems. The platform enables easy deployment of applications by exposing to developers a service oriented interface to access things (according to a Things-as-a-Service model) regardless of the technology and the physical infrastructure they belong to.

Keywords IoT platforms · M2M · Local cloud · Fog computing · Context-awareness

The final publication is available at Springer via http://dx.doi.org/10.1007/s11277-015-2639-0

1 Introduction

The recent advancements in embedded computing and sensor technologies are turning the Internet of Things into reality. Many solutions commercially available today exploit networked smart objects to provide end-users with advanced services connected to the *physical* world. Such solutions are however often *vertical*, isolated, systems based on ad-hoc HW/SW realizations which are not able to cooperate with each other to share common smart object capabilities. Isolation is not the only drawback: from a software developer perspective, the lack of a common software fabric to interact with smart objects entails great limitations on software portability and maintenance [1].

To overcome such limitations, a layered *horizontal* approach is by far more appropriate and desirable, since it eases the integration of heterogeneous existing systems, and also facilitates the development of IoT applications based on a unified interface to a converged infrastructure. In fact, several horizontal IoT platforms have been recently designed and developed exposing standard inter-faces to access smart objects. Most of these solutions are characterized by a centralized cloud-based approach, which yields the usual benefits in terms of scalability (potentially infinite computation and storage capacity), ease of maintenance, time to market and low development costs. On the other hand, running an IoT platform in a cloud infrastructure deployed far from where the smart objects are physically located may result in a sub-optimal choice for many classes of IoT applications, e.g., Machine-to-Machine (M2M) ones, which typically have a limited scope in time and space (data need to be processed only when and where it is generated), require simple and repetitive closed-loop interactions, and often must respond with stringent latency guarantees to avoid service disruption.

As a practical example, consider a confined environment, like a smart home, where a number of M2M systems are already deployed, such as an *alarm* system equipped with presence sensors for surveillance, an *environmental control* system including temperature sensors for heating and cooling control as well as light switch actuators, and a *garden watering* system with humidity sensors. In this scenario, one might want to deploy and run an all-in-one IoT platform to enable the development of new applications leveraging sensors and actuators from all the available M2M systems. Solutions relying on centralized cloud

storage and computational capabilities would obviously re-quire continuous connection and data offloading towards remote external systems, even to support applications that only need to exploit data generated locally. Moreover, a centralized architecture fails to support applications that require proximity to physical deployments. As an example, an ap-plication for home security would very much increase its effectiveness at a minimum cost if, in case of an alarm event, it could also directly control the smart lights available at home as part of a different system. This however requires real-time interaction with the latter and accurate context collection, which can be guaranteed only by a fully local deployment.

BETaaS, Building the Environment for the Things-as-a-Service a European project funded under the 7th Framework Programme aims at overcoming such limitations through the creation of a horizontal runtime platform. A distinctive and novel feature of the BETaaS platform is its architecture, which is based on a distributed runtime environment made of a so-called *local cloud* of nodes that allows accessing smart objects connected to the platform regardless of their technology and physical location. The distributed environment can be installed on devices such as network gateways, home routers, set-top boxes, etc., characterized by heterogeneous storage and computational capabilities, generically hereafter referred shortly as *gateways*. Such architecture does not only en-able the deployment of private/isolated platforms but also allows applications to run close to the IoT physical deployments, with a scope that can span over different domains by means of direct interaction among gateways. The proximity between applications and smart objects is of paramount importance as a crucial enabler for M2M applications that rely on timed and fresh information from smart objects.

On top of the distributed runtime environment, the BETaaS platform provides a unified framework to the software developer of M2M applications through a content-centric service-oriented interface, named Thing-as-a-Service. Such unified programming interface, mandatory to cut software development time and enable code re-usability, is exposed to applications to interact with smart objects with the support of **semantic technologies** and regardless of their location, technology or communication protocol. The platform is designed with a modular structure that supports **integration** and **expandability**. This structure facilitates the integration of existing vertical M2M systems to allow, on the one hand, applications to interact with things from different environments, and, on the other hand, to preserve implementations and functionalities of existing solutions. Such modular architecture allows also easy **customization** that is supported by allowing the development of custom services (named *extended services*) running on the platform.

In addition, the BETaaS platform provides built-in support for several non-functional requirements. These extra functionalities are included to support the wide variety of application environments envisaged for future IoT platforms [3]. Although a subset of them are sporadically offered by existing platforms, to the best of our knowledge BETaaS is the first one that provides applications with all the following functionalities by design:

- **Context Awareness** is implemented by means of semantic support for discovery and thing selection by means of efficient and scalable context identification and management.
- **Quality of Service** (QoS) is supported for applications that require timed interaction with physical objects.
- **Security** is included to secure access to sensitive data and to improve the platform management through things and gateways trust assessments.
- **Big Data Management** is provided to handle the large amount of information generated by things and to offer applications big data functionalities.
- **Virtualization** is finally included to guarantee an efficient management of storage and computation resources and isolation between different applications that share the same horizontal platform.

In this paper we first present the concept underlying the BETaaS project, and then we provide an overview of the platform, recently released as open-source product. The remainder of the paper is structured as follows. In Section II an overview of the related work is presented, Section III introduces the BETaaS concepts and overview the platform architecture, in Section IV we present the interface offered to software developers to build applications for the BETaaS platform, Section V provides some details of the open-source platform implementation, finally in Section VI we draw the conclusions.

2 Related Work

Several initiatives have been carried out towards the definition of horizontal platforms for the development of IoT applications. Differently from the BETaaS approach, the majority of such initiatives adopts a centralized architecture. The *OpenIoT* project [1], for instance, exploits cloud computing to run an open-source middleware that supports the creation of services offering virtualization of IoT devices and context awareness. The *ClouT* project [2] adopts a cloud-based approach as well, however their efforts are specifically tailored to the smart-city use case. The *COMPOSE* project [3], instead, focuses on mobile applications to enable easy access to things through a cloud-based architecture for integration and scalability. Finally, a number of open horizontal cloud-based platforms are also commercially available, such as, for example, *Xively* [4], originated by the former infrastructure named *Pachube*.

More recently, a novel approach that goes beyond traditional centralized computing towards a geographically distributed architecture has been proposed. This is referred as *Fog Computing* [6], which fosters the move of computational and storage capabilities to the *edge* of the network. Although such

[1] http://openiot.eu/
[2] http://clout-project.eu/
[3] http://www.compose-project.eu/
[4] http://xively.com/

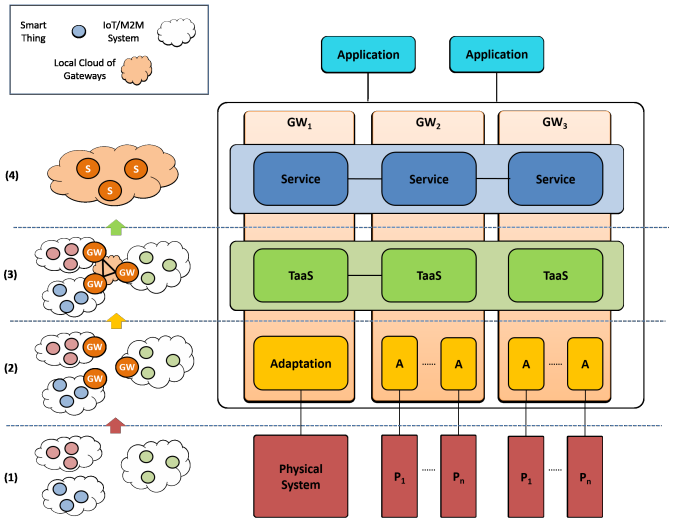

Fig. 1 BETaaS concept and architecture.

approach is recognized as the long-term evolution to support M2M applications [7], only a few decentralized solutions have been proposed in literature so far. However, they are usually bounded to a specific technology, e.g. [8], that exploits the CoAP protocol, or provide only a basic set of functionalities to applications, e.g. [9], that focuses only on interoperability and integration.

Regarding standardization, it is worth to mention the work of the *oneM2M* consortium [5], which aims at developing technical specifications of a common Service Layer to rely upon interconnecting M2M devices. Finally, among the many research projects on the field it is worth to mention the *IoT-A* project [6] that defined an architectural reference model and an initial set of key building blocks to foster the future IoT, though no actual implementation has been provided.

3 BETaaS Concept and Architecture

The BETaaS concept and reference architecture are illustrated in Figure 1. The logical steps bringing forth a converged M2M service platform starting from a number of already existing systems are shown from bottom to top on the left side of the picture. More specifically, M2M systems composed of different smart objects (1) are integrated through a set of gateways, each one providing access to its locally connected M2M system (2). Gateways then cluster together through a tight mutual interaction so as to logically form a **local cloud** of gateways, which provides the distributed runtime environment hosting the BETaaS platform (3). The term "local cloud" referring to the set

[5] http://www.onem2m.org/
[6] http://www.iot-a.eu/

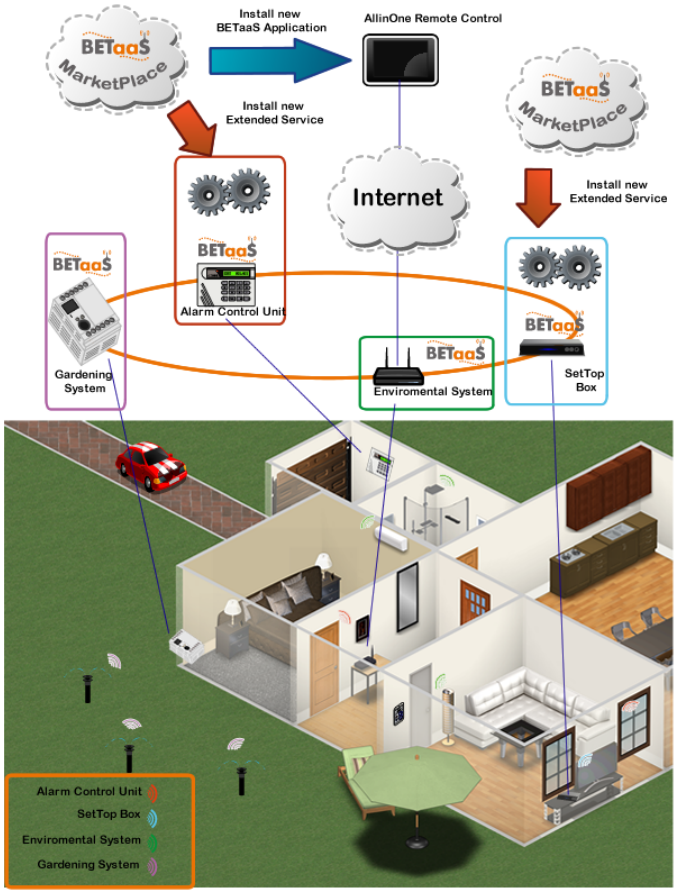

Fig. 2 BETaaS platform Use-Case example.

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, i.e.,

- *Resource pooling*. An application cannot know/control which physical device will provide the required service. Efficient service selection based on context information is implemented to optimize the usage of resources.
- *Rapid elasticity*. The distributed nature of the architecture makes it highly scalable and suitable to handle bursts of requests.
- *Measured service*. Execution of services is based on the current status of physical devices hosting the gateways, which is therefore monitored to implement resource optimization.

On top of the local cloud of gateways, a number of extended services can be further deployed (4), in addition to basic ones, in order to expose to appli-

cations the underlying sensing and actuating infrastructure as a service. The BETaaS reference architecture (on the right side of Figure 1) reflects the logical steps previously described by means of a corresponding layered structure. This structure guarantees the proper level of abstraction to applications at the top layer, and the flexibility needed to integrate different systems characterized by heterogeneous technologies at the bottom layer. Transparent integration of existing systems is achieved through platform-side adapters installed at the *adaptation* layer which provides a uniform interface to the layer above and converts requests from the latter to access existing systems according to their respective technology. On top of the adaptation layer there is the *TaaS* layer, which is at the core of the BETaaS platform. This layer implements the *Things-as-a-Service* model, which defines a common interface to access things as a service regardless of their specific technology, communication protocol and location. For each smart thing, one or more *thing service(s)* are derived in a content-centric manner and exposed to applications. In order to provide also abstraction from physical location, i.e., allow applications to transparently access thing services irrespectively of the gateway, the TaaS layer is implemented in a distributed fashion to cooperatively share resources among gateways, thus realizing the concept of local cloud previously introduced. We refer the reader to our previous work [2] for a detailed description of the functional view of the BETaaS architecture.

On top of the TaaS layer, the platform implements the service layer, which defines the interface to external applications. By default, this layer exposes all thing services available in a running instance of the platform as *basic services* that enable applications to interact directly with smart objects. In addition, the service layer also allows the dynamic deployment of custom services, named *extended services*, possibly developed by a third party. Extended services can be used to extend the functionalities of the platform by implementing complex logic tailored to a specific running instance, or can be exploited by applications to push functionalities, such as real-time control, close to the platform, where they can run independently of the status of the remote application. In-platform deployment of extended services does not only allow the platform to support a wide range of application environments through customization, but also opens the possibility to establish a **digital market** of extended services that can be installed dynamically by end users on demand.

Figure 2 shows BETaaS potential usage in the smart-home use case previously presented. The alarm, environmental and irrigation systems are integrated into an all-inclusive runtime environment and a unified standard interface is exposed to developers that can create applications leveraging resources from all systems in a seamless manner. In order to enable this integration, a number of BETaaS gateways need to be installed. This can be performed through a combination of (*i*) installing additional hardware that is natively running the BETaaS software and connects to the M2M system through a dedicated adapter; and (*ii*) modifying whenever possible the original system in order to implement all or a subset of layer functionalities of the BETaaS platform, e.g., through an update of the software running on the control unit.

While the existing systems preserve their original functionalities, new enhanced ones are now enabled by BETaaS. For instance, an extended service can be installed on the platform to enhance the environmental control system by exploiting humidity and presence information available from the others, e.g., turning down the air-conditioning when a window is opened as results from the magnetic sensor part of the alarm system. Applications interacting with the platform and running on external devices, instead, can be installed on a smartphone to expose a uniform control interface to end users. An all-in-one application running on a smartphone, for example, can offer users the remote control of all the systems according to users preference and context information, e.g., by turning on/off the heating/cooling system depending on whether the user is approaching/leaving the house.

Finally, it is worth to highlight that as a matter of fact the complete separation between applications and the sensing and actuating infrastructure realized by the TaaS layer allows third-party developers to build generic applications which can be run in any instance of the BETaaS platform. This is key to enabling the development of an M2M application market from where BETaaS applications/extended services can be downloaded and installed by users in their own platforms.

4 Building Software for the BETaaS Platform

BETaaS provides software developers with two types of mechanisms to allow exploiting the resources managed by the platform. The first one consists of developing external applications with their own logic that access the services exposed by the platform. In the second case, BETaaS is dynamically extended through the installation of custom services. On the resource side, BETaaS provides mechanisms for an M2M system to be seamlessly integrated into the platform through a transparent interface. Finally, a set of enhanced capabilities is natively provided to applications/extended services to ease their development by focusing on the implementation of the application logic.

4.1 Application programming interface

The BETaaS platform exposes services to applications through both Web Services and RESTful APIs. The set of available operations is quite small so that no complex interaction is required:

- *Installation*, an application requests the platform to allocate the resources needed for its execution;
- *Service invocation*, applications request/send data from/to one basic or extended service;
- *Registration*, an application register to a basic or extended service to receive data notifications;
- *Notification*, the platform notifies to applications new available data;

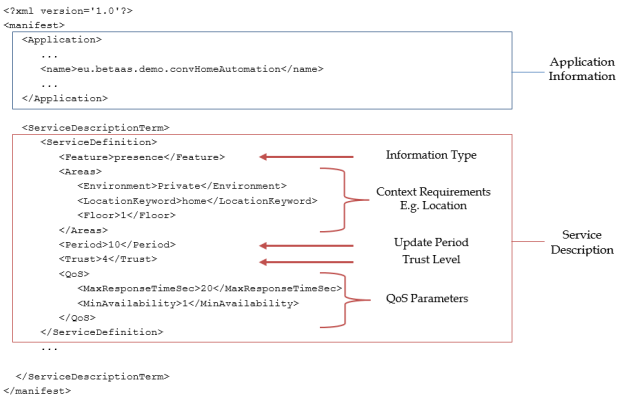

Fig. 3 Simplified example of application manifest.

The *installation* procedure requires a set of structured information about the resources to be allocated. Applications pass such information to BETaaS through a *manifest* document, as illustrated in Figure 3, which contains specifications on the required service through its semantic description through a set of natural language keywords. In particular the manifest contains the required information type, e.g. presence information, and the required context attributes, e.g. the location of interest. To complete the service description, other additional requirements might be specified, such as the security level, the QoS and the trust level. Given this description, the platform takes care of selecting and aggregating the necessary thing services in order to match the manifest description. Easy of usage and flexibility are the main strengths of this approach, which allows developers to describe the required service through a description, which is easy to understand and modify.

As the installation procedure terminates successfully, the application can request/send data from/to one basic or extended service specified in the manifest. In order to support a wide range of applications, the platform defines two different methods to invoke basic or extended services: through single service invocation or through registration/notifications. Through single service *invocation* an application requests/sends data from/to a service one-time, e.g., to retrieve the current value from a sensor. Through *registration/notification*, instead, an application registers its interest to the platform in unsolicited updates on the value of a service. As the value of a given service changes, the platform takes care of notifying all the applications that registered to the service with the new available data.

4.2 Extended Service support

BETaaS allows developers and vendors to implement extended services to integrate custom services. The flexible framework that BETaaS is based on allows

their dynamic deployment as software bundles. Extended services operate just like applications and are suitable for delivering complete solutions including application logic. Extended services may operate as automatic processes (e.g., to implement closed-loop control logic as a true M2M application) or can also expose an interface to applications.

4.3 Integration of existing systems

Integration of an existing M2M system can be performed transparently through the definition of an *Adaptation Layer*. The major capabilities of each Adaptation Layer are as follows:

1. ability to automatically detect, connect and communicate with the underlying device infrastructure,
2. gathering and creation of contextual information based on the information provided by the hardware (or software) of the physical device
3. collect missing contextual information through supplementary configuration documents that complete the semantic profile within the platform represented by a software component called Thing object.

This object holds the necessary information and when forwarded to the upper layers, creates services for handling the devices not as simple hardware components but as smart objects. In parallel, the Adaptation Layer is responsible for accommodating low level support available to the layers above such as the periodic notification of values/metrics of particular things to subscribed services, which are notified on periods described at subscription time.

4.4 Platform capabilities

The BETaaS platform implements by design a set of enhanced capabilities that are available to applications and are briefly described in the following.

4.4.1 Context Management

BETaaS is a context aware platform, which means that it is aware of the circumstances that may affect the behavior of the smart objects connected to it. Among the circumstances that BETaaS considers we have concepts such as the location of smart objects or the type of feature offered by them. Through context management BETaaS is able to: (a) unify information coming from heterogeneous resources in the physical environment, as information collected by the Adaptation Layer is modeled by an ontology, i.e., the BETaaS ontology, (b) generate unique thing service names for each of the smart objects, and (c) infer new knowledge from raw data in a context-aware fashion. Knowledge inference is performed by two different mechanisms. On the one hand, we have defined semantic rules based on the BETaaS scenarios, e.g., we have defined a

rule to detect equivalent thing services, which are those associated to things of the same time in the same location. On the other hand, we have defined two word-sense disambiguation (WSD) algorithms to infer information from locations and from sensor/actuator types. Information is inferred both when a smart object is attached, and when an application demands information, e.g., if an application demands the temperature at home, a temperature sensor installed in the kitchen is valid (kitchen is meronym of home).

Every gateway in the platform stores a BETaaS ontology. The BETaaS ontology is a network of ontologies that we have created reusing ontologies that are relevant in their domains and that model the BETaaS scenarios. In order to promote standardization, the BETaaS ontology is populated whenever possible with the common vocabulary provided by a lexical database that groups English words into sets of synonyms or *synsets*. Such database is based on the semantic relationships between synsets (hypernymy, hyponymy, holonymy, meronymy). All synsets inserted in the BETaaS ontology are stored following these relationships.

4.4.2 Quality of Service

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 [4]. Since at run-time applications can only select one class with a fixed set 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-stage procedure: 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 simplicity, application developers specify the QoS parameters required for each service directly into the Manifest file. Developers of extended services can exploit, instead, an advanced Service Level Negotiation interface to allow complex developments. For the sake of interoperability, a standard protocol is exposed by the TaaS layer: the WS-Agreement Negotiation protocol [5], which is the de-facto standard for SLA agreement negotiation, establishment and management for Web Services.

In order to enforce and monitor the negotiated QoS requirements, a QoS framework has been defined and implemented in the platform in order to

ensure an efficient management of resources to optimize their usage and guarantee the fulfillment of the agreements. The framework included in BETaaS is based on a two-phase procedure, namely, *reservation* and *allocation*. The reservation phase is handled by a *Broker*. The Broker manages the QoS negotiation, performs admission control and, most importantly, manages resource reservation. The allocation phase, instead, is managed by a *Dispatcher*. The Dispatcher performs allocation of resources at time of invocation. Resource allocation can manage the resources following different optimization goals. In the current release an algorithm that optimizes the energy efficiency of battery powered smart-sensors has been implemented as described in [10].

For an exhaustive description of the QoS framework, the interested reader can refer to [11].

4.4.3 Security Management

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 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. After mapping each service specified in the manifest to the required thing services, the platform evaluates the access policies. As a result, a set of tokens is granted to the application. Every time a service is invoked, the token is verified confirming access rights and conditions.

Relying on external systems manufactured and maintained by independent third-parties can raise trust issues. For this reason BETaaS 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.

4.4.4 Big Data Management

In order to handle the amount of data generated by the things in large deployments such as a smart city scenario, a big data manager that exploits the BETaaS distributed architecture is included. A flexible data distribution paradigm that allows gateways to distribute data from local things to remote services and provides a storage service (through a SQL database) is adopted. These data services overcome the limitation of a single gateway, where resource could be limited. Moreover, more databases services could be deployed in a BETaaS instance, so that a failure or the unavailability of a local data service does not compromise the whole data layer.

The big data manager connects these database services to a Big Data platform for the purpose of storing and analytics processing of large amount of data. Loading tasks from a SQL DMBS to a distributed file system are performed at regular intervals, keeping local database data growth limited to only recent data. The big data platform leverages a metastore to define a structure for the data loaded into the distributed file system, leveraging a Presto DB to offer clients a SQL interface to access the stored data.

The big data capabilities of the platform are exposed to applications through a specific module of the big data manager, which provides an interface, named *data task*, which allows the analytics deployment, i.e. queries run by PrestoDB that returns results to client applications. Such interface provides a mechanism to describe the input parameters that are used to specify the query and to represent the returned data. The module responsible for managing the data task has different capabilities: it provides the list of the available data tasks that applications could run to perform analytics and controls the applications access to tasks and used resources.

4.4.5 Virtualization

Virtualization capabilities are 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, based on a very lightweight Linux operating system. In case of big data, there is an image for computation nodes and another image for storage nodes, while in the case of applications deployment, there is a Java web container for deploying web applications.

It is possible to instantiate the same image with different resources, depending on the requirements. The algorithm to allocate resources minimizes fragmentation by taking into account the resources required by the existing VMs and the potential requirements for new instances. The priority is to exploit those resources provided by local gateways but, in the case not enough resources are available, the algorithm will try to use an external Cloud (if external accounts have been configured).

Application developers may provide information about the resources required by their applications in the Manifest file used during the installation, in an OVF-like format, so the platform can allocate resources accordingly.

5 Platform Implementation

The first release of the BETaaS platform, including a substantial subset of the platform capabilities, has been released as open-source software on github [7]. This implementation is based on the OSGi [8] 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 bundle groups by using features. A feature is actually a list of bundles with their related dependencies, which can then be 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.

The modularity offered by OSGi is exploited to guarantee the expandability of the platform, i.e., additional "extended" services can be deployed as third-party bundles at runtime, with a full automatic management of their lifecycle. For the same reason, platform capabilities are implemented as bundles as follows.

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. The BETaaS network of ontologies has been created using used the following ontologies: SSN [9], Time [10], CF [11], Phenonet [12], MUO [13], FIPA [14] and GeoNames [15]. Ontology development has been performed through Apache Jena [16].

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

[7] https://github.com/BETaaS
[8] http://www.osgi.org
[9] http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628
[10] http://www.w3.org/2006/time
[11] http://www.w3.org/2005/Incubator/ssn/ssnx/cf/cf-property
[12] http://www.w3.org/2005/Incubator/ssn/ssnx/meteo/phenonet
[13] http://purl.oclc.org/NET/muo/muo
[14] http://www.fipa.org/specs/fipa00091/PC00091A.html
[15] http://www.geonames.org/ontology/ontology_v3.1.rdf
[16] https://jena.apache.org/

Parser of the gateway. In order to promote standardization, the Semantic Parser uses WordNet [17] to translate this contextual information to WordNet synsets whenever possible, and stores this data in the ontology.

WordNet organization is based on semantic relationships between synsets (hypernymy, hyponymy, holonymy and meronymy). All synsets inserted in the BETaaS ontology are stored following these relationships, through SKOS [18]. The relationships between the terms are used as a mechanism of knowledge inference: e.g. if an application demands the temperature at home, a temperature sensor installed in the kitchen is valid (kitchen is meronym of home).

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 [19]. The implementation of access condition within the capability or token is done based on the Condition field in the XACML (eXtended Access Control Markup Language) standard, using a library called JBossXACML [20].

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

For an exhaustive description of the platform implementation details, we refer the interested reader to the public deliverables available on the project website.

[17] http://wordnetweb.princeton.edu
[18] http://www.w3.org/2004/02/skos/intro
[19] https://www.bouncycastle.org/java.html
[20] http://picketbox.jboss.org/

In addition to closed laboratory tests performed periodically to check module integration and proper basic software functionalities, the platform is validated through two field trials set in two different scenarios: smart home and smart city. In the smart-home trial, an existing proprietary domotic system is integrated into the platform to demonstrate how existing closed systems can be successfully integrated. In the smart city trial, planned to test platform scalability on large-scale and test features like QoS and Big Data, a smart parking system is deployed to help drivers finding a parking spot using the presence sensors installed on lamp posts, and to optimize the car distribution, using traffic information. In this scenario, the M2M systems integrated in the platform are based on an open standard, i.e., the ETSI M2M and the Constrained Application Protocol ($CoAP$). For a more detailed description of the trials and a presentation of the experiment results we refer the interested reader to the deliverables publicily available on the project website [21].

6 Conclusion

In this paper we have presented the BETaaS platform, an open-source runtime platform for the execution of M2M applications that facilitates the integration of existing IoT systems, and provides software developers with a high-level, content-centric, abstraction to access smart objects resources, along with a built-in support for several non-functional requirements. The platform leverages on a distributed architecture made of gateways that are interconnected through the BETaaS software thus forming local cloud of gateways on which M2M applications can run exploiting a unified interface to interact with smart-objects. We believe that the BETaaS open-source framework will play an important role in facilitating the creation of open IoT systems and on which third-party M2M applications can run, breaking the barriers that are refraining the expansion of the IoT market.

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 a project co-founded by the European Commission under the 7th Framework Programme (grant no. 317674).

References

1. Rellermeyer, Jan S., et al. The software fabric for the internet of things. In The Internet of Things, Springer Berlin Heidelberg, 2008.
2. E. Mingozzi, G. Tanganelli, C. Vallati, V. Di Gregorio, An Open Framework for Accessing Things as a Service in Proceedings of the 16th International Symposium on Wireless Personal Multimedia Communications (WPMC 2013), Atlantic City, NJ, USA, June 24-27, 2013.

[21] http://www.betaas.eu/deliverables.html

3. Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 29, 7 (September 2013)
4. Nef, Marie-Aurlie, et al. Enabling QoS in the Internet of Things. CTRQ 2012, The Fifth International Conference on Communication Theory, Reliability, and Quality of Service.
5. Oliver Waeldrich, et al, Web Services Agreement Negotiation Specification, WS-Agreement Negotiation.
6. Bonomi, et al. Fog computing and its role in the internet of things MCC '12.
7. Abdelwahab, at al. Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler, IEEE Internet of Things Journal, 2014.
8. Mainetti, et al. Discovery and Mash-up of Physical Resources through a Web of Things Architecture. Journal of Communications Software & Systems, 2014.
9. Sarkar, et al. A scalable distributed architecture towards unifying IoT applications, WF-IoT, 2014
10. 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.
11. 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.