Abstract

This report contains the definition of the architecture for the BETaaS platform, including all the components in the platform and the main external interfaces, in order to clarify how interactions will happen among the components. It also gives some initial hints about the potential deployment of the components. This architecture will be used as the base of the implementation and updated in M16, according to testing and validation results.
# Document History

<table>
<thead>
<tr>
<th>Vers.</th>
<th>Issue Date</th>
<th>Total pages</th>
<th>Modified pages</th>
<th>Modifications implemented</th>
<th>Author</th>
<th>Organisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>25/03/2013</td>
<td>12</td>
<td>-</td>
<td>ToC, introduction and section 3.1, initial architecture in section 4</td>
<td>Fco. Javier Nieto</td>
<td>ATOS</td>
</tr>
<tr>
<td>0.2</td>
<td>20/05/2013</td>
<td>14</td>
<td>6</td>
<td>Section 5.4, updates to section 4 and 3</td>
<td>Fco. Javier Nieto, Sergio García</td>
<td>ATOS</td>
</tr>
<tr>
<td>0.3</td>
<td>20/06/2013</td>
<td>22</td>
<td>9</td>
<td>Content for section 3.7, 4.2, 5.6, 3.4, 5.5, 3.6, 5.7, 6</td>
<td>ATOS, TECN, INTECS</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>26/06/2013</td>
<td>30</td>
<td>15</td>
<td>Updates in BigData, QoSM, Things Adaptor (sections 3 and 5), Updates to the architecture (section 4)</td>
<td>ATOS, HP, CONV</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>27/06/2013</td>
<td>32</td>
<td>13</td>
<td>Updates in 5.1, 5.2, 5.3, 5.5, 5.6 and 5.7</td>
<td>ATOS, TECN, INTECS</td>
<td></td>
</tr>
<tr>
<td>0.6</td>
<td>28/06/2013</td>
<td>36</td>
<td>10</td>
<td>Updates of diagrams and addition information about interfaces in section 5</td>
<td>ATOS, TECN, CONV, HP</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>29/06/2013</td>
<td>41</td>
<td>10</td>
<td>Updates in section 1, section 3 and section 6</td>
<td>Fco. Javier Nieto</td>
<td>ATOS</td>
</tr>
<tr>
<td>1.1</td>
<td>30/06/2013</td>
<td>41</td>
<td>3</td>
<td>Minor updates in several sections</td>
<td>Fco. Javier Nieto, Izaskun Mendia, Belén Martínez</td>
<td>ATOS, TECN</td>
</tr>
</tbody>
</table>
Executive Summary

According to the BETaaS conceptual model and the definition of the basic and extended capabilities, BETaaS must cover several functionalities at different levels: adaptation, TaaS and service. In this document, we have analysed the required capabilities, extracting a list of functionalities to be provided by BETaaS platforms. These functionalities have been described and have been mapped with the capabilities at the different layers. Based on these functionalities, a high level architecture has been defined in such a way it can cover all the functionalities and, at the same time, it does not require to perform isolated implementations for each layer. All those components identified have been detailed with concrete designs, in order to facilitate their implementation and, moreover, their external interfaces have been identified, in order to facilitate the definition of the interactions in the API focused document. Finally, deployment options have been described, in order to provide a clear idea on how to configure BETaaS gateways.
# Table of Contents

**Executive Summary** ............................................................................................................. 3

1. **Introduction** ...................................................................................................................... 6

2. **Table of acronyms** ............................................................................................................ 7

3. **Capabilities and Functionalities** ..................................................................................... 8
   3.1. BETaaS Capabilities ........................................................................................................... 8
   3.2. Things Access Adaptation ............................................................................................... 8
   3.3. Resources Management and Discovery ............................................................................ 9
   3.4. Security Services .............................................................................................................. 10
   3.5. QoS Services ................................................................................................................... 10
   3.6. Big Data Management .................................................................................................... 11
   3.7. Context Management ..................................................................................................... 14
   3.8. Services and Applications Management .......................................................................... 14
      3.8.1. Application Execution .............................................................................................. 15
   3.9. Instance Management ..................................................................................................... 15
      3.9.1. GW Join ................................................................................................................... 15
      3.9.2. GW Lifecycle .......................................................................................................... 16

4. **Global Architecture** ........................................................................................................ 18
   4.1. High Level Architecture ................................................................................................. 18
   4.2. Components Description ................................................................................................ 18
   4.3. High Level Functionalities .............................................................................................. 19
      4.3.1. Register new gateway in an Instance ....................................................................... 19
      4.3.2. Register things ......................................................................................................... 20
      4.3.3. Install application .................................................................................................... 21

5. **Components Detail** .......................................................................................................... 23
   5.1. Instance Manager ............................................................................................................ 23
      5.1.1. Component Architecture .......................................................................................... 23
      5.1.2. Package Provided External Interfaces ........................................................................ 23
      5.1.3. Package Required External Interfaces ....................................................................... 23
   5.2. Service Manager .............................................................................................................. 23
      5.2.1. Component Architecture .......................................................................................... 23
      5.2.2. Package Provided External Interfaces ........................................................................ 23
      5.2.3. Package Required External Interfaces ....................................................................... 23
   5.3. TaaS Resources Manager ............................................................................................... 25
      5.3.1. Component Architecture .......................................................................................... 25
      5.3.2. Package Provided External Interfaces ........................................................................ 25
      5.3.3. Package Required External Interfaces ....................................................................... 25
   5.4. QoS Manager ................................................................................................................... 26
      5.4.1. Component Architecture .......................................................................................... 26
      5.4.2. Package Provided External Interfaces ........................................................................ 26
      5.4.3. Package Required External Interfaces ....................................................................... 26
   5.5. Security Manager ............................................................................................................ 28
      5.5.1. Component Architecture .......................................................................................... 28
      5.5.2. Dependability Manager ............................................................................................ 28
5.5.2.1. Package Provided External Interfaces ................................................. 29
5.5.2.2. Package Required External Interfaces ............................................. 29
5.5.3. Trust Manager ................................................................................. 29
  5.5.3.1. Package Provided External Interfaces ............................................... 30
  5.5.3.2. Package Required External Interfaces ............................................. 31
5.5.4. Package Provided External Interfaces .................................................. 31
5.5.5. Package Required External Interfaces .................................................. 31

5.6. Context Manager .................................................................................. 31
  5.6.1. Component Architecture ................................................................. 31
  5.6.2. Package Provided External Interfaces ............................................... 32
  5.6.3. Package Required External Interfaces ............................................. 33

5.7. Big Data Manager ................................................................................. 33
  5.7.1. Component Architecture ................................................................. 33
  5.7.2. Package Provided External Interfaces ............................................... 35
  5.7.3. Package Required External Interfaces ............................................. 35

5.8. VMs Manager ....................................................................................... 36
  5.8.1. Component Architecture ................................................................. 36
  5.8.2. Package Provided External Interfaces ............................................... 36
  5.8.3. Package Required External Interfaces ............................................. 37

5.9. Things Adaptor ...................................................................................... 37
  5.9.1. Component Architecture ................................................................. 37
  5.9.2. Package Provided External Interfaces ............................................... 37
  5.9.3. Package Required External Interfaces ............................................. 38

6. Platform Deployment ............................................................................... 39

7. Conclusions ............................................................................................. 41

8. References ............................................................................................... 42
1. Introduction

Once a conceptual model and the basic and extended capabilities of the BETaaS platform have been defined, together with some use cases, it is possible to analyse the architecture to be followed during the implementation. In this document we analyse the outcomes of those models and capabilities in order to extract the final functionalities to be provided by BETaaS instances. The functionalities have been grouped for facilitating the definition of the architecture.

Based on the functionalities identified, the document presents a high level architecture, valid for implementing those steps described in the use cases in a coherent way. Since a high level architecture is not specific enough, the documents defines in detail each component of the architecture, detailing subcomponents and interfaces for interacting.

Section 3 of this document presents the capabilities identified in the conceptual model, together with the functionalities which can be mapped. Some of the functionalities and concepts are detailed more in deep in this section as well, in order to clarify the base of the architecture. Section 4 describes the high level architecture and its components. Moreover, it provides details about the interactions performed in several situations belonging to the use cases defined in the project. Section 5 provides a detailed design of each component in the architecture, with component diagrams, and a description of the external interfaces. In section 6 we clarify how the components of the architecture can be deployed in gateways. Finally, section 7 presents some conclusions related to the architecture.
### 2. Table of acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Expanded form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>Application Layer</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APP</td>
<td>Application</td>
</tr>
<tr>
<td>BD</td>
<td>Big Data</td>
</tr>
<tr>
<td>BDM</td>
<td>Big Data Manager</td>
</tr>
<tr>
<td>BETaaS</td>
<td>Building the Environment for the Things as a Service</td>
</tr>
<tr>
<td>CM</td>
<td>Context Manager</td>
</tr>
<tr>
<td>CoAP</td>
<td>Constrained Application Protocol</td>
</tr>
<tr>
<td>CoAPTA</td>
<td>CoAP Things Adaptor</td>
</tr>
<tr>
<td>CoAPPHY</td>
<td>CoAP Physical Layer</td>
</tr>
<tr>
<td>CPU</td>
<td>Computational Power Unit</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>GW</td>
<td>Gateway</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IM</td>
<td>Instance Manager</td>
</tr>
<tr>
<td>IoC</td>
<td>Internet of Contents</td>
</tr>
<tr>
<td>IoS</td>
<td>Internet of Services</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine to Machine</td>
</tr>
<tr>
<td>OCCI</td>
<td>Open Cloud Computing Interface</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>QoSM</td>
<td>Quality of Service Manager</td>
</tr>
<tr>
<td>SCM</td>
<td>Security Manager</td>
</tr>
<tr>
<td>SM</td>
<td>Service Manager</td>
</tr>
<tr>
<td>TaaS</td>
<td>Thing as a Service</td>
</tr>
<tr>
<td>TaaSRM</td>
<td>Resource Manager (TaaS Layer)</td>
</tr>
<tr>
<td>TA</td>
<td>Things Adaptor</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VMM</td>
<td>Virtual Machine Manager</td>
</tr>
</tbody>
</table>
3. Capabilities and Functionalities

3.1. BETaaS Capabilities

The BETaaS capabilities have been defined in those deliverables obtained as result of previous tasks (T1.4 and T2.1) [1][2]. These capabilities are grouped in four layers: Physical, Adaptation, TaaS and Service. Some of these capabilities have been also detailed in [3], providing more indications on how they are performed. The following figures show the list of capabilities identified so far.

**Figure 1: BETaaS layer capabilities.**

BETaaS will cover those capabilities related to Adaptation, TaaS and Service, focusing especially on those listed for TaaS and Service.

3.2. Things Access Adaptation

This covers those functionalities related to the access to things and the adaptations to be done, so things can be used and managed correctly in BETaaS. There will be one adaptation per each kind of physical platform (i.e. OSGi).

- In general adaptation of things starts with the discovery of physical devices as they exists within the BETaaS “cloud”. This is mostly a responsibility of the Physical layer but also of the Adaptation layer which has to make the communication with the things and also provide the means to services and application for communicating and using the things according to their needs.

- Access thing. Further to discovery and once initial communication is established, management of the Things is an outcome of accessing them via a well-organized identification process and monitoring their state. So an authorization process is based on the unique identity of the Things upon which access to the Thing is guaranteed from within the BETaaS instance (security of the service or application which uses eventually this Thing is not discussed here).

- Monitor thing. Using the access to the Things BETaaS adaptation layer is responsible for providing the state and overall situation of the Things.
3.3. Resources Management and Discovery

It is necessary to provide functionalities for managing all the resources available in a BETaaS instance, including things and other kind of computational resources provided by gateways (i.e. storage, CPU, etc…). Those services and applications being executed in a BETaaS instances will provide some requirements related to what they need. Depending on the functionalities to cover, they may need a concrete list of sensors and actuators. Appropriate resources will be allocated depending on those requirements, but taking also into account the characteristics of the Things Services available, as well as security and quality concerns.

![Diagram of BETaaS resources management](image)

**Figure 2: BETaaS resources management.**

Once those resources are allocated, they are continuously monitored when services and applications are executed. Other applications and services (existing or new ones) will require more resources, so it is necessary to have optimization mechanisms which will take into account potential re-allocation of resources, in order to support the execution of more services and applications, as a way to improve scalability and increase reliability.

In order to cover these needs, the functionalities related to resources management are the following:

- **Discover resources**: Available things will be discovered, so new thing services will be generated in order to be used in future services and application executions.

- **Add/update/remove resources to registry**: There will be a registry for managing those resources available in the instance, existing coordination between gateways. They will be added as they are discovered, updated as they change and removed as they become unavailable.

- **Get resource data**: When required, data about a resource will be provided, not only about the way to access it, but also about other characteristics.

- **Perform action on resource**: It will be possible to perform actions on those known and controlled resources. In case of sensors, for instance, this means to change their configuration and retrieve data, while in the case of actuators, this would be to activate an action (i.e., turn on a light).

- **Provide resources allocation**: When an application or service needs a set of resources, there will be an analysis of the application needs and the available resources (their capabilities, status, trust, QoS, dependability, etc…), so the most adequate resources will be assigned to the application.

- **Re-allocate resources**: The conditions of the environment where the BETaaS instance is located change as new applications and users want to use those services provided by the instance, or new gateways and things join the instance. In such cases, the platform will be able to adapt the resources allocated to services and applications, in order to maintain an optimized context.
3.4. Security Services

Security modules are part of most layers of the BETaaS architecture, except the adaptation one. Each layer is burdened with the responsibility to retrieve and impose a particular security policy (or policies).

- Calculate resources trust: Retrieve and calculate each resource (physical entity) trusts. This is done in the TaaS layer mainly, since it will calculate trust for things services. This calculation will be the base for calculating later on trust about applications and services.
- Calculate services/applications trust: Retrieve and calculate each application or service trust. This is performed on the top Service layer, using as input the trust calculated for things services which are being used by applications and services.
- Get security policy: Impose a particular security policy as taken from input to the BETaaS instance. This functionality must allow retrieving information about those existing policies (not only the policy itself, but also how many times it has been applied and in which conditions).
- Apply security policy throughout the application and physical layer. As there will be several policies affecting different layers, BETaaS instances must monitor the environment and apply the policies.
- Add/modify/remove security policy depending on the changed input to the BETaaS instance or the client application/BETaaS instance. In the end, this comprises those tasks for managing policies, as needed.
- Validate certificate of the connecting application or BETaaS instance. It is necessary to guarantee that client applications are not malicious and their identity must verified through certificates.
- Manage things identities: The adaptation layer is also responsible for gathering and managing the identities of the BETaaS instance physical devices which have to be uniquely named in order to proceed with servicing the upper layers of the architecture.

There are functionalities which will be used on demand (i.e., those related to authentication), while others will be working all the time in order to guarantee BETaaS instances are secure enough all the time. Moreover, we must take into account that the outcome of some security functionalities will be the input for others, such as in the case of policies application, in which certificates validation and trust calculation may influence the activation of policies. These functionalities have been further explained in [3].

3.5. QoS Services

As defined in [3], QoS is one of the extended capabilities of BETaaS. It must be managed at different levels, mainly at the service and at the TaaS layer. The functionalities which map with this management are:

- Negotiate QoS conditions: Given that an application or service is installed in the instance, there is a negotiation of QoS conditions, in order to understand the conditions that the application requires and the conditions which things services can provide. Depending on the result, it is possible to obtain a ranking.
- Monitor QoS: When applications and services are executed in an instance, QoS is continuously monitored, in order to guarantee that agreements done are fulfilled.
- Calculate QoS: QoS is calculated at the service level and at the TaaS level, in order to support QoS negotiations and QoS monitoring. The model applied will depend on the element under evaluation.

Basically, the first stage, when installing applications, is to perform a negotiation of the QoS conditions which can be offered by the thing services available. This way, a ranking is provided, in order to facilitate the selection of the most adequate things. In the process, the QoS is calculated depending on different aspects.

Then, when an application is going to be executed, the allocation of resources is performed in a definitive way, and that is the moment in which the QoS monitoring will be started. In case there is any agreement not fulfilled, thanks to this monitoring, it is possible to request recovery actions.
These processes require interactions not only with several components in the same gateway, but also with components in other gateways, since it may happen that an application is using things services implemented in different gateways. The whole approach is explained in detail at [3].

3.6. Big Data Management

Data management is an important features of BETaaS instances; since a lot of data is generated by sensors and consumed by services, its growth might became an issues that affect the overall platform performance, moreover storage and elaboration of large amount of data requires designs of appropriate technologies to be placed inside the system.

In BETaaS the data generation come from sensors; for this reason already at TaaS level there is a BETaaS component capable of performing simple tasks over the data such as: collecting data from sensors, performing some basic adaptation of data structure and delivering it to a dedicate storage area. Once the data has been gathered from sensor to a more appropriate storage area, even temporarily, an issue should arise: what to do when this data grow. For managing this issue, every gateway can store information or be configured to deliver data to another gateway where additional storage functionalities are available; in this particular gateway, BIG DATA functionalities are enabled. This special gateway does not necessary contains a TaaS layer, but it runs some fundamental BETaaS service layer components that allow to: receive, store and process large amount of data. This gateway not only receives data collected by different gateways, it also process data by running specific application queries and jobs over the stored data. The storage of data is based on a distributed file system, so that replication, parallelism and high availability is guaranteed.

BETaaS's big data capabilities can be summarized in:

- Gather data from source
- Store data
- Process data

The first important functionality is related to gathering information from the source, the things, to the final storage area. If a gateway can generate data, meaning that is able to collect information from the resources, then a component that collects this data is required. At a TaaS level, the big data component is the one used for this purpose. It provides three basic TaaS data capabilities:

- Collection
- Adapt
- Load

Collection component is essential for determining when to collect the information and how. In the TaaS BD component is responsible for establishing when the data is available so that it can collect it. For example it can actively request data at regular interval or data can be retrieved upon generation from the things. In this way there is a certain degree of flexibility for the collection of data on the system to adapt to different scenarios, this enable to keep in consideration resources available and save energy consumption for devices with limited capabilities.
Figure 3 BETaaS’s gathered data flows from the things to the final distributed FS area.

Adaptation is an optional capability, meaning that is not required for each TaaS BD component to have it installed, in fact constraints of resources available can force the system to deliver data postponing later additional processing. In any case at TaaS level processing can be only minimal and necessary to avoid loss of information. For example it might be required in order to adapt the data generated by heterogeneous things in a common format, or can be used to deliver data with the same time interval or unit of measure.

Load is another essential capability of TaaS BD component: without it the data would not be delivered in the final storage system. This component schedules the delivery of data to a Service layer BD component. Information can be delivered to a service component belonging to the same gateway or one in another gateway. In fact, to avoid useless load for small system with limited capability, the delivery can be routed to a more adapt gateway without the necessity of having a load service BD component in each gateway. The delivery of data can be actively done by the Load component of the TaaS BD or can be polled remotely by the service BD that wants to retrieve the information.

Figure 4 BETaaS TaaS data capabilities

As the Figure 4 shows, gateway 1 (with limited resource) deliver its data, collected from the things, to a service component of another gateway; this component receives data from both TaaS on Gateway 1 and Gateway 2. In this case the service component acts as collector and overcome the limited resources of Gateway 1. Also different functionalities are deployed for Gateway 1, it only collects and loads information, while Gateway 2 also performs simple adaptations to its information before loading them on to the service layer.
Together with the BD at TaaS layer, there is another essential component that allows BETaaS instances to gather data; this is the BD at service layer, its features are:

- Receiving
- Transforming
- Delivering
- Storing
- Processing

Receiving, as the name says, is the feature that receives or retrieves information from one or more TaaS. As said before, it can perform this actively or passively. After it is received, data is sent to additional transformation steps or sent to another gateway if in the current gateway there is no data storage.

Transformation inside this component is made to ensure that its format adapts to storage and later processing. Transformation at service level can involve data collected from different gateways and thus require some homogenization tasks.

Deliver is a capability that a BD service component uses only if the current gateway is not the final destination of the data. The final repository of data is where all information is stored for BD processing purpose. Deliver component has the purpose of delivering the information to a gateway where the data will be stored.

Store is a capability that connects the service BD component to a storage system, controlled by the gateway. Typically this component connects to a distributed file system. The BETaaS BD service component, by using storage functionalities, is capable of storing information inside the distributed file system accessible to the gateway. It is present only on gateway where the resources are capable of storing large amount of data.

Process component is used to perform or scheduled processing of data for the purpose of providing results to the applications. It contains the available processing jobs, the schedule of active jobs and manages the execution of the data process. It is also responsible for taking allowing application to perform real time query when resources are available.

The BETaaS BD service, by using the analytic functionalities, allows application to perform operation on the large amount of data collected. There are two patterns of using this data: real time query and batch processing. The real time processing is useful for application that performs query of recent data and need an immediate response; the application provides parameters and the component perform a query on the data available to the application. The batch processing, instead, consists in a set of available processing jobs not initially enabled, but that can be requested by an application. These process are over a large amount of data and for these reason need to be scheduled carefully, a batch processing job is enabled by the BD component only when an application request it and it schedule it so that it does not impact the available resources of the gateway.

In the Figure 5, two different gateways are depicted. The gateway 2 deliver the information received to a BD gateway, where resources are available. The BD gateway takes the information received from other gateway 2 and applies the necessary transformations, after these steps are terminated, data is stored on a distributed file system. At the same level, another component, Application Data Service, allows application to process and retrieve these data is installed.
Figure 6 shows a gateway where the Big Data capabilities are enabled. The gateway by using its storage capabilities, store the data inside a distributed file system. This is where the instance data are gathered and allowed to grow up over the time. When a user application ask for information, the Application Data Service detects what kind of job is required to be run on the data, then it can enable it, or it can directly perform a real time query on a small subset of available data.

3.7. Context Management

BETaaS will be a content-centric platform that will semantically process the information retrieved from the things connected to it. The context that surrounds these things will also be taken into account in order to provide the more suitable services at all times. The management of all the semantic processes that take part in BETaaS will be done by the Context Manager. This component will provide the following functionalities:

- Translation of the information retrieved/sent from/to the things into a semantic format. This information will be mapped into an ontology.
- Managing the context or circumstances in which things that are connected to BETaaS are embedded. This context information will be mapped into a context ontology.
- Managing of the subscriptions/unsubscriptions made by the applications that use BETaaS to the elements in the ontology.
- Providing of an interface with the BETaaS ontologies, to insert or retrieve information from them.
- Generation of the thing services used to expose the things as services.
- Generation of the services to be provided to the applications.
- Sending the Big Data Manager all the information about the Things registered in BETaaS, together with the data that those Things are providing.

3.8. Services and Applications Management

The main task of this functionality is to allow BETaaS applications to use the BETaaS platform in order to access data provided by devices. Since services and applications are executed in BETaaS, it is necessary to provide functionalities covering the requested capabilities:

- Check required resources
- Add/modify/remove/run/stop service and application
- Find service and application
- Get service and application
- Request resources to lower levels

This capability allows the application to exploit the distributed architecture by managing services installed remotely on different GWs.

The architecture comprises a GW with main role that is called GW*. GW* resides at the center of a star topology configuration and mainly manages GWs registration and services look-up and access.

Here following an example of typical scenario involving this capability.

### 3.8.1. Application Execution

The application attached to GW1, requires a service (the service is installed in GW2).

The SM of GW1 checks if the service is configured locally; if not the SM communicates with the SM* that keeps the global view of the instance and knows the correspondence between GW and installed services. SM* knows that the service is installed in GW2 by invoking its Service Registry and forwards the request to the SM2. If Qos and authorization parameters are fulfilled, the SM2 can access the ThingServices provided by the lower level.

![Application Execution Diagram](image)

#### Figure 7: Application Execution

### 3.9. Instance Management

The InstanceManager is responsible for managing information about the configuration and status of GW joined in the BETaaS Instance. Furthermore it negotiates the security and trust policy access between GWs and BETaaS Instance. The instance management consist on providing those functionalities required so gateways can join or leave BETaaS instances.

Here following an example of typical scenario involving this capability.

#### 3.9.1. GW Join

GW1 wants to join the BETaaS Instance. The InstanceManager invokes the join interface provided by GW* that checks if the new GW has the required permissions by invoking the SecureManager. In case all the constraints are fulfilled, GW* updates its Configuration Registry.

Once the GW has joined the BETaaS Instance, it needs to publish its own services. GW1 invokes the publish interfaces provided by the SM*. SM* invokes IM* in order to verify that the GW1, has been already included in the BETaaS Instance. If the procedure ends successfully, GW* updates the Service Registry containing the reference to the GW1 where the services are installed.
3.9.2. GW Lifecycle

Gateways can work in different ways depending on their status. The lifecycle of the gateways include different status for the gateways and how a gateway can change its status, depending on the context and the needs of the BETaaS instance which it belongs to.

Initially, a gateway is turned off so, once it is turned on, the first status of the gateway is Stand-alone. In this status, the gateway belongs to its own BETaaS instance, managing the resources and the instance (it is the one used as connection point for new gateways to join the instance and it centralizes certain interactions). In that status, it is the only gateway belonging to the instance.

If the gateway finds another BETaaS instance and it wants to join it, it will go to the In-Instance status if it is accepted by the instance. In this case, it does not manage the BETaaS instance, but it provides its resources and services to the instance and it can use those resources and services provided by other gateways.

It may happen that a gateway is in the Stand-alone status and another external gateway asks to join the instance the gateway is managing. In that case, the gateway would go to the α-Mode status, since it will keep managing the BETaaS instance.

When a gateway is part of an instance and it is not the gateway controlling the instance, it is not in alpha mode but, in case something happens with the gateway in alpha mode (it needs more resources, it has some errors or it was disconnected suddenly), a gateway can be promoted to be the instance administrator. In this case, the gateway would go from In-Instance mode to the α-Mode status.
A gateway can abandon the α-Mode status because of different reasons (i.e. it may need to leave some functionalities, so its resources will be used for other purposes). As a BETaaS instance always needs to have a gateway managing the instance, the gateway has to leave the management by delegating the management to other gateway, so it will be just another gateway in the instance, moving to In-Instance status.

Gateways which are part of an instance can decide to leave it for different reasons. In this situation, the gateway would go from In-Instance status to Stand-alone status again.

Finally, gateways in In-Instance or Stand-alone status can go to Turn-off status, so the gateway can be shut down without negative effects for the instance (i.e. not removing things and services without notifying to others).
4. Global Architecture

4.1. High Level Architecture

The BETaaS architecture has been designed in such a way that it covers all the functionalities required and it is modular so, in some cases, for certain gateways it will be possible to avoid the deployment and usage of some components.

![BETaaS High Level Architecture](image)

Figure 10: BETaaS high level architecture.

The figure shows the main components and the relationships. The two main components are the TaaS ResourcesManager and the ThingsAdaptor, since they represent the things and the way to manage them for providing the functionalities required.

4.2. Components Description

This subsection is for describing with enough detail the components included in the global architecture, so it will possible to understand their purpose and their role in the platform.

- **InstancesManager**: This component is in charge of those interactions to be performed between the BETaaS Instance and the rest of the world. It provides the means to registering new gateways and managing the BETaaS Instance, enabling interaction between gateways.

- **ServicesManager**: This is the component which acts as the interface for managing services and applications in the BETaaS Platform. It will gather all the needed information about the services and applications, so all the requested resources can be allocated while, at the same time, its requirements about security and QoS are fulfilled. It needs to interact with the TaaSResourcesManager for allocating the resources (things, devices, other resources...).

- **TaaSResourcesManager**: This is a component which is in the center of the architecture. It decides how to manage the available resources (things, storage, CPU, etc...). It gathers all the information available in the platform in order to take the best decision. This includes QoS evaluation and prediction, trust and security, semantic context information, data about the things and any other input available from the analysis of Big Data which may help. It requests actions to the VMManager (in case certain computational resources are needed) and the ThingsAdaptor for requesting operations on them.

- **ThingsAdaptor**: This is an important component, since it represents the bridge between the things and the BETaaS capabilities related to the things. It is able to discover things connected to the BETaaS instance, maintain data about them, monitor them, access them and provide those adaptation interfaces so other components can provide other capabilities (QoS, resources...).
management, trust, context management). It also communicates directly to things, since it will contain those drivers needed for supporting different solutions (i.e. ZigBee...). It needs to provide, not only the network protocols, but also other potential needs depending on the driver used (i.e. identifiers management). It unifies the information received from heterogeneous things (different manufacturers, different standards, etc.) connected to the BETaaS instance. The component transforms the raw data emitted by the things into an agreed format like XML. It makes use of the ContextManager to parse this predefined format into a semantic format that is mapped into the BETaaS ontologies.

- **BigDataManager**: This component provides advanced storage capabilities for large amounts of data coming from things and their monitoring. It also performs data analysis in order to find more information about the data usage and other data processing which can be done. It can take advantage of the VMManager in case it needs to use VMs for special storage purposes, when the capability is available.

- **VMManager**: In those gateways in which this is possible, this component will create, remove, start and stop lightweight VMs which can be used for performing processing operations or providing isolated storage. It will use those APIs provided by a hypervisor running in the gateway OS (in the case of BETaaS, we plan to use Xen) for performing these operations and for retrieving some monitoring data about the performance of these VMs, so any potential issues may be detected and corrected.

- **QoSManager**: This component retrieves information about the BETaaS instance and determines the QoS level which is currently being offered to those BETaaS services and applications using things. It checks that QoS levels are fulfilled and maintains QoS policies which may influence the resources management performed in the BETaaS instance.

- **ContextManager**: This is the component which manages all the semantic processes in BETaaS. The ContextManager offers an interface to the ThingsAdaptor, which invokes the ContextManager to translate the data that has been emitted by things (and that has been pre-processed by the ThingsAdaptor) to a semantic format that is mapped into the BETaaS ontologies. On the other hand, the ContextManager invokes the ThingsAdaptor when a value has to be sent to an actuator. In this case the ContextManager translates the information from a semantic format to a format that the ThingsAdaptor is able to process. The ContextManager also offers an interface to the TaasResourceManager. The component accepts petitions from the TaasResourceManager to set values to actuators, and sends information to the TaasResourceManager when a new measurement has been taken by a sensor (in this case, an application has had to explicitly ask for this information).

- **SecurityManager**: It is necessary to provide some security related capabilities. This component provides these services, including management of security policies, trust calculation and other basic services used at different levels (validation of certificates, identities management, encryption/decryption of data to be sent over networks, etc.). Depending on the operation to perform, it may need information about things, their context, QoS issues and provided levels, etc...

### 4.3. High Level Functionalities

This subsection contains some UML sequence diagrams according to the functionalities defined in section 3. These sequence diagrams show high level interactions between the platform components in order to provide a concrete functionality. We have used as reference the Home Automation scenario as defined in [4], detailing the interactions among components which should be carried out in certain steps.

#### 4.3.1. Register new gateway in an Instance

When registering a new gateway in an existing BETaaS instance (in this case, a private one), it is necessary to perform some interaction between different gateways. Once a gateway decides to join a new instance (this could be forced by a instance owner, through a GUI, for instance), the Instance Manager at the new gateway interacts with the ‘alpha’ gateway of the instance, where another Instance Manager will manage all the process for joining.

In case the Instance Manager accepts the new gateway, several interactions need to happen between components in the new gateway and the Service Manager and the Taas Resource Manager in the
‘alpha’ gateway. These interactions are, basically, for synchronizing services and things services provided by the new gateway and the instance, so both parts will have a complete vision about the resources and services available in the whole instance.

Figure 11: Register a new gateway in an instance.

4.3.2. Register things

One of the steps in the use case requires registering new things in the BETaaS instance. The Instance Manager will start the process, since it provides a GUI from which users could trigger the process.

The Instance Manager will request to the Things Adaptor initialize the discovery of new things in the range of the instance. In this case, the example shows the discovery process based on the CoAP (Constrained Application Protocol) protocol. A component in the Things Adaptor side, contacts with a component in the physical layer, obtaining the list of new things found.

Figure 12: Register things.

The Context Manager is notified about the new things available, so it can evaluate again the things services to be generated and used. These new thing services (or updated ones) are sent to the TaaS
Resources Manager, so it will keep them in an internal catalogue and will be able to re-allocate resources.

At the same time, there are some components which need to monitor and evaluate the behaviour of the new things found. The QoS Manager will analyse if the new things achieve certain QoS levels. The Security Manager will evaluate trust and validate any certificate provided. Finally, the Big Data Manager is implied because it is the component storing and managing the monitoring information.

4.3.3. Install application

When a new application is installed, it is necessary to follow a process for allocating the necessary resources, in order to guarantee the BETaaS instance is able to execute the application in the required terms.

A user will request to install a new application, so the Service Manager will start the process. The first step is to determine if the necessary resources can be allocated, taking into account non-functional requirements and checking that the required things services are available as well. Then, the application is registered for starting its execution, confirming that those resources reserved will be used in a short time.

![Diagram showing the steps of installing an application]

**Figure 13: Install application.**

The Service Manager asks the TaaS Resource Manager to allocate resources. The TaaS Resource Manager, requests to the Context Manager updated information about available things services. Since there are some non-functional requirements, the Security Manager is contacted for obtaining a ranking
of things services sorted by the trust result, and the QoS Manager is requested for a ranking sorted by
the QoS levels provided. Then, the TaaS Resource Manager selects the preliminary list of thing
services to use and the connections with things are checked, in order to guarantee all the things we
need are accessible.

Then, the Service Manager registers the application, in order to start the execution of the application.
The TaaS Resources Manager receives the request and registers the application in the Security
Manager and the QoS Manager, so they will be aware of the final allocation of resources and will start
monitoring the behavior of the things services selected. Finally, the context in the Context Manager is
updated, so the next time a resources is going to be used, there will be a trace of the current usage of
each resource, and decisions taken will take into account the current things load.
5. Components Detail

5.1. Instance Manager

5.1.1. Component Architecture

This component is composed of two different sub-components:

- Configuration
- GWsManager

![Instance Manager Component Diagram](image)

**Figure 14: Instance Manager Component**

The Configuration is in charge of managing the local GW parameters like the IP address, GW* address and the status of the devices. In case the component belongs to GW* it manages all references to GWs within the same instance.

GWsManager is responsible for managing the communication between BETaaS Instance and a GW that wants to join it. This component provides two interfaces towards ServicesManager component and, in case of GW*, towards InstanceManager of joining GWs.

5.1.2. Package Provided External Interfaces

- **IF_InstanceManager::CheckJoinGW** – it is used by the ServiceManager in order to verify that the specified GW has been registered in the Betaas Instance. This interface is provided only by GW*: **IF_InstanceManager::GWJoin** - this interface allows the joining procedure of a new GW into the Betaas Instance. The GW* exposes this interface allowing GWs to use it by means of the corresponding component InstanceManager.

5.1.3. Package Required External Interfaces

- **IF_SecurityManager** – it is used by InstanceManager to check if new GW has the permissions to join the BETaaS Instance.

5.2. Service Manager

5.2.1. Component Architecture

The ServiceManager comprises three sub-components:

- **ApplicationCore** (Consumer)
This module is in charge of managing the application requests. It communicates with the ServiceCore in order to use the requested service.

- **ServiceCore (Provider)**
  This module receives the request from the Consumer and checks if the specific service is properly configured in the local Registry; if so, it checks the QoS and authorization parameters. In case all constraints are fulfilled, the info acquired from the Registry, are used in order to access the service. Results of service execution are returned to the ApplicationCore.

In case the Service is not configured locally, the Provider forwards the requests to the corresponding Provider of GW*.

This component communicates with the lower level when TaaS creates a new ThingService in order to register it in the Local Registry.

During installation of application, the TaaS is invoked to allocate the necessary resources.

Furthermore, the Provider of the GW* invokes its Local IM in order to verify that the GWs, that want to publish their services, have been already included in the BETaaS Instance.

- **Registry**
  This module contains all the information related to services (for instance, URL, resources, etc); in case of GW*, the Registry contains the reference to the GWs where the services are installed.

---

**Figure 15: Service Manager Component**

### 5.2.2. Package Provided External Interfaces

- **IF_ServiceManager::Run** – it is used by application in order to run the service on BetaaS and to get the results. It is the only interface used by an actor external to BETaaS Instance.

- **IF_ServiceManager::RunService** – it is used by the ServiceManager to request GW* to run a service not present in the local GW. The interface is provided only by GW*.

- **IF_ServiceManager::PublishServices** – it is used by the ServiceManager of joining GW to publish the list of its own services to GW*. The interface is provided only by GW*.

- **IF_ServiceManager::RegisterThingsService** – it is used by TaasLayer in order to register new ThingsServices.
5.2.3. Package Required External Interfaces

- **IF_QoSManager** it is used in order to check that the QoS requirements are fulfilled.
- **IF_SecurityManager** – it is used in order to check if the application has got the rights to use the services.
- **IF_TaaS::AllocateResorces** - It is used during application installation in order to request Taas to allocate the necessary resources.
- **IF_InstanceManager::CheckJoinGW**: it is used by the GW* in order to verify that the specified GW has joined the BETaaS Instance. It is invoked during services publication.

5.3. TaaS Resources Manager

5.3.1. Component Architecture

The TaaSResourceManager is one of the main components, since it has the responsibility of managing the resources of the gateway, allocating them to services depending on their needs. Its architecture is shown in the following figure.

![Figure 16: TaaS Resource Manager Component](image)

The ResourcesOptimizer will receive the requests for allocating resources. It will analyse the list of available resources, they load, the QoS conditions, security conditions, etc… in order to determine if the service can be accepted and reserve the resources to be used for that service.

The ResourcesCatalogue will contain information about Thing Services available and will be updating it continuously, so Thing Services endpoints can be built and the ResourcesOptimizer will be able to retrieve data about them. On the other hand, the ServicesCatalogue will provide information about the services provided, including a mapping with the allocated resources.

The EndpointsManager is in charge of generating the endpoints for Thing Services. It will use the input from the ResourcesCatalogue and the support of the ContextManager in order to determine methods and parameters to be used when accessing Thing Services.

Finally, TaaSResourceClients will implement the clients for enabling interactions with other components in the same gateway (i.e. Security Manager, QoS Manager, VM Manager, BD Manager…), as well as with the TaaSResourceManager located in other gateways.

5.3.2. Package Provided External Interfaces

- **IF_TaaSRM::RegisterThingsService** – This is used for registering new Things Services when they are created.
- **IF_TaaSRM::RemoveThingsService** – It is used for removing Things Services, as they are not available anymore.
- **IF_TaaSRM::Synchronize** – It is used to synchronize with other TaaSRMs in the same BETaaS instance (i.e. when joining an instance).
- **IF_TaaSRM::RegisterApplication** – This method is for registering a new application.
- **IF_TaaSRM::UnregisterApplication** – It is for removing any data from an existing application.
- **IF_TaaSRM::FreeLocalResources** – This is for releasing resources associated to applications or services, once they are not necessary anymore.
- **IF_TaaSRM::AllocateResources** – It allocates the required resources to applications and services.

### 5.3.3. Package Required External Interfaces

- **IF_CM** – It is used for retrieving semantic data about Things Services from the Context Manager.
- **IF_ExtTaaSRM** – it is used for interacting with other TaaSRMs at the same BETaaS instance, but located in other gateways.
- **IF_IntGW** – This interface is for interacting with other components in the same gateway: Security Manager, BD Manager, QoS Manager, VM Manager and Things Adaptor.

### 5.4. QoS Manager

#### 5.4.1. Component Architecture

The QoSManager is in charge of all the QoS related operations performed. Since these operations are related to capabilities at different levels of the conceptual model, the internal component architecture must show this separation, in order to enable flexibility and increase evolution capabilities.

![QoSManager Diagram](image)

**Figure 17: Detailed design for the QoSManager.**

All the components must be aware that there are several ways of calculating QoS and adapting it to those things which are part of a BETaaS Instance. The idea is to enable a ‘Strategy’ pattern, so the core classes of each component may fulfill required APIs and functionalities and, at the same time, adapt their behavior to the underlying infrastructure. We take also into consideration that the interactions among components at the same layer (but different Gateways) differs when QoS is determined at the Service layer and at the TaaS layer, for instance.
Another important point to take into account is the monitoring of those Service Level Agreements (SLAs) done after negotiations. As we expect to provide a certain QoS level, we will have monitoring components which will be comparing obtained quality against that quality promised during the negotiation. Since this is different also at each level (because of different parameters will be monitored), ServiceQoSMonitor and TaaSQoSMonitor will check if the quality levels promised are met.

Due to the difference of QoS usage among layers, we need to specify more in detail the internal architecture of each subcomponent. In this case, in line with the specifications of the extended capabilities [ref!!], we include the required components for enabling the interactions among components at the same layer and for calculating the specific QoS for the layer.

![Figure 18: Design for the TaaSQoSManager subcomponent.](image)

In this case, as shown in the figure above, the TaaSQoSManager uses a QoSBroker and a QoSDispatcher in order to perform negotiation and to generate a ranking at the TaaS layer. They will make use of the result generated by the TaaSQoSCalculator to determine if negotiations succeed and to sort out the ranking to be provided as result.

![Figure 19: Design for the TaaSQoSCalculator subcomponent.](image)

Finally, the previous figure shows an example of one of the subcomponents (TaaSQoSCalculator) applying the strategy pattern. Depending on the context, the TQoSCalculator will select the kind of calculator and will create the corresponding instance, accessing to the functionalities through a standardized API.

### 5.4.2. Package Provided External Interfaces

- **IF_SQoS::RegisterApplicationQoS** – This method is used for confirming the resources allocation of an application, so the QoS will be correctly monitored.
- **IF_SQoS::UnregisterApplicationQoS** – It is for indicating that the resources of an application have been released (since it has finished its execution), so there is no need to monitor it.
- **IF_SQoS::getServiceQoS** – Depending on the Things Services allocated, it is possible to obtain the current QoS value for a service or application.
- **IF_TaaSQoS::getThingsQoSLevel** – It provides the current QoS level of Things Services to be used.
- **IF_TaaSQoS::writeThingServicesQoS** – When new Things Services are registered, the QoS Manager must be notified, so it will be able to monitor them and calculate QoS.
- **IF_TaaSQoS::modifyThingServicesQoS** – It provides information about changes in the Things Services, so the QoS calculation can be adapted to the new context.

### 5.4.3. Package Required External Interfaces
- **IF_TA** – This interface is for connecting with the Things Adaptor, in order to retrieve certain information about them.
- **IF_ExtQoS** – During QoS negotiation, internal subcomponents need to interact with other QoS Managers located in other gateways. This interface will be used for that purpose.
- **IF_BDM** – It is for retrieving monitoring information stored in and managed by the Big Data Manager.

### 5.5. Security Manager

#### 5.5.1. Component Architecture

There are several aspects to be taken into account for providing security. Authentication (and certainly authorization) can be implemented on an overall module that provides the means for identification, authentication and finally authorization. Moreover the general Security Manager is responsible for the management and tracking of the Certification keys between different types of entities, thus the **SecurityBroker** (figure below) should provide interfaces to all different layers of the architecture. By collaborating with the TrustManager the decision is made for Authorization of the client for the requested access. In case an error is received from the **DependabilityManager** the SecurityBroker will provide the necessary actions.

![SecurityManager Architecture Diagram](image)

**Figure 20: Design for the SecurityManager component.**

In the following sections, the **DependabilityManager** and the **TrustManager** are specified, since they represent some of the most important extended capabilities in BETaaS.

#### 5.5.2. Dependability Manager

The main aspect of dependability comprises the failures/errors management; in particular the solution aims to firstly catch the errors, then identify the possible causes and finally select and act the proper recovery actions.
The Monitoring Module receives failures notifications from modules that have identified some problems within any layer. Furthermore the Monitoring module is polled by the supervisor in order to check if the Gateway is properly monitored.

The SuperVisorMonitoring is invoked by the Monitoring and requires self-analysis on each GW by means of the corresponding Local Diagnosis Module. The Global Diagnosis Module collects the analysis results from each GW and the Planner decides the recovery action to be performed by the RecoveryModule through by TaasRM.

5.5.2.1. Package Provided External Interfaces

- IF_DependabilityManger::FailureNotification – it is used by any module catching a failure in order to notify it to the system.

5.5.2.2. Package Required External interfaces

- IF_TaasRM::RecoveryActions – it is invoked by the Dependability Manager after the failure analysis results in order to put in action the identified recovery procedure (restart some component, force GW into a safe state).

5.5.3. Trust Manager

This subcomponent is in charge of calculating trust for different conceptual layers: service layer and TaaS layer. It retrieves information from several sources and applies customized trust models, depending on the layer. This requires interacting with several components at different layers and at different gateways also.

**Figure 21: Dependability Manager**

The Monitoring Module receives failures notifications from modules that have identified some problems within any layer. Furthermore the Monitoring module is polled by the supervisor in order to check if the Gateway is properly monitored.

The SuperVisorMonitoring is invoked by the Monitoring and requires self-analysis on each GW by means of the corresponding Local Diagnosis Module. The Global Diagnosis Module collects the analysis results from each GW and the Planner decides the recovery action to be performed by the RecoveryModule through by TaasRM.

5.5.2.1. Package Provided External Interfaces

- IF_DependabilityManger::FailureNotification – it is used by any module catching a failure in order to notify it to the system.

5.5.2.2. Package Required External interfaces

- IF_TaasRM::RecoveryActions – it is invoked by the Dependability Manager after the failure analysis results in order to put in action the identified recovery procedure (restart some component, force GW into a safe state).

5.5.3. Trust Manager

This subcomponent is in charge of calculating trust for different conceptual layers: service layer and TaaS layer. It retrieves information from several sources and applies customized trust models, depending on the layer. This requires interacting with several components at different layers and at different gateways also.
According to the previous figure, ServiceTrustManager component will take care of the trust at the service layer, while TaaSTrustManager will do so with the TaaS layer. As both of them require to interact with many components, they share a ClientsManager, which will manage all these interactions, acting as some kind of façade which will facilitate the update of clients, being transparent to those components calculating trust.

If we take a look into detail, TaaSTrustManager and ServiceTrustManager are very similar. Both of them will follow an internal architecture as shown in the next figure.

In this case, the TaaSTrustCalculator calculates individual trust aspects, retrieving the data it needs through the TaaSProxy. Once these calculations are finished, the TaaSAggregator will apply the fuzzy model for obtaining the final trust result. Since trust needs to be calculated periodically and it could require to check and notify trust thresholds, the TaaSThread will be the component forcing trust calculation periodically and performing trust checks, in order to raise alarms whenever necessary.

### 5.5.3.1 Package Provided External Interfaces

- **IF_TaaSTrustManager::GetTrust** – Any component can use this method for obtaining the current trust value calculated for a Things Service. It can also provide a ranking of Thing Services sorted by the trust value.

- **IF_TaaSTrustManager::RegisterThingsService** – it is used by Taas Layer in order to register new Things Services. It indicates trust for a new Things Service has to be calculated periodically.
5.5.3.2. Package Required External Interfaces

- **IF_BDM** – This interface is used in order to retrieve monitoring data or any other information provided by the BDM.
- **IF_ExtTM** – It is used for interacting with another Trust Manager located in other gateway.
- **IF_QoSM** – It is used for retrieving information from the QoS Manager.

5.5.4. Package Provided External Interfaces

Those interfaces provided by the Trust Manager and the Dependability Manager are also provided by the Security Manager. Moreover, several interfaces are added, related to other functionalities provided by this component:

- **IF_SM::authenticate** – It is used to perform authentication for different elements which will connect to a BETaaS instance.
- **IF_SM::checkAuthorization** – It checks that certain things services and/or services are authorized to perform certain operations.
- **IF_SM::getSecurityAspects** – This method provides information about security aspects for things services and services.

5.5.5. Package Required External Interfaces

The required external interfaces for this component are only those already defined in the Dependability Manager and the Trust Manager. No new external interfaces are required.

5.6. Context Manager

5.6.1. Component Architecture

The Context Manager controls all the processes related to semantics in BETaaS. Except from the SemanticParserAdaptor, which belongs to the Adaptation Layer of the BETaaS functional model, the rest of the subcomponents of the Context Manager are part of the TaaS Layer.

The Context Manager offers an interface to the ThingsAdaptor through the SemanticParserAdaptor. This subcomponent is responsible for translating into a semantic format the data that is emitted or received by the things connected to BETaaS. Once the information emitted by a thing has been processed by the ThingsAdaptor, this component invokes the SemanticParserAdaptor which translates the data into a semantic format that is mapped into the BETaaS ontologies. A Things Ontology will store all the information related to the measurements that the things connected to a gateway take. The
context associated to those things will be stored in a Context Ontology. Both ontologies will be part of the TaaS Layer of the gateway.

![Diagram of Context Manager and related components]

Figure 24: Detailed design for the ContextManager.

The Context Manager generates the thing services, which is the way in which the TaaS Layer exposes basic services which can be mapped directly to one or more things. When a thing is connected to BETaaS, the ThingServicesManager generates both the thing service and its associated thing-service-id, which like the rest of the information related to things, is stored into the BETaaS ontologies.

The Context Manager also generates the services to be used by applications. These services are associated to the thing services, though this relationship is not one to one. In fact, there are cases in which one service can be provided by several thing services, the same as one thing service can be provided by several things. When one Thing is disconnected from BETaaS, the CM analyzes if this disconnection implies the deletion of any thing service, and or the stopping of any service.

The TaaSRM of the gateway will use the ThingServicesManager as an interface with the ontologies and with the rest of subcomponents of the Context manager, which means that all the communication between the TaaSRM and the Context Manager will be done by invoking thing services.

A Subscription Manager will manage the subscriptions that applications may create in some elements of the ontologies. This mechanism will allow BETaaS to inform applications when any change in the status of the elements to which are subscribed happen.

The Context Manager will send the Big Data Manager all the information about the Things that are registered in BETaaS and the data that these Things are providing. A rule programmed in the Things Ontology will send a BigData Agent all the new information collected about the Things and their data. This BigData Agent will send all the collected information to the BETaaS Big Data Manager.

5.6.2. Package Provided External Interfaces

- IF_TeamsAdaptor::translateThingsInformation(thingsData) – it is used by the ThingsAdaptor to ask the CM to translate the information about the Things into a semantic format.
• IF_ThingsAdaptor::removeThings(thingsData) – it is used by the ThingsAdaptor to ask the CM to remove the information about the Things that have been disconnected from BETaaS.

• IF_TaaSRM::getContextThingServices(thingServicesList) – it is used by the TaaSRM to ask the CM the Thing Services that suit the context needed for the resources of an application.

• IF_TaaSRM::subscribe(thingsList) – it is used by the TaaSRM to tell the CM to subscribe to the selected Things in the corresponding BETaaS ontologies.

• IF_TaaSRM::getRealTimeInformation(thingsList) – it is used by the TaaSRM to tell the CM to ask for real time information about the selected Things.

• IF_TaaSRM::setActuatorsValues(thingsList,valuesList) – it is used by the TaaSRM to send the CM the new values of the actuators.

• IF_TaaSRM::unsubscribe(thingsList) – it is used by the TaaSRM to tell the CM to unsubscribe to the selected Things in the corresponding BETaaS ontologies.

5.6.3. Package Required External Interfaces

• IF_ThingsAdaptor::getRealTimeAdaptedInformation(thingsData) – it is used by the CM to ask the ThingsAdaptor for real time information about the selected Things.

• IF_ThingsAdaptor::setActuatorsAdaptedValues(thingsData) – it is used by the CM to send the ThingsAdaptor the new value of an actuator (in a pre-defined format like XML).

• IF_TaaSRM::notifyNewMeasurement(appID,thingID,data) – it is used by the CM to notify the TaaSRM about the new measurement made by a sensor.

• IF_TaaSRM::registerThingServices([thingID,thingServicesList,servicesList]) – it is used by the CM to inform the TaaSRM about the Things, and the associated Thing Services and services that have been generated by the CM.

• IF_TaaSRM::delete([thingID,thingServicesList,servicesList]) – it is used by the CM to tell the TaaSRM to delete the Things that have been disconnected, and to delete the associated Thing Services and /or to stop the corresponding Services, if it is the case.

• IF_BDM::setThingsBDM(thingsList[thingID,data]) – it is used by the CM to tell the BDM the information about the Things and their data.

5.7. Big Data Manager

5.7.1. Component Architecture

Management of data occurs in both TaaS and Service layer, even if with different purposes. BD at TaaS layer has the purpose of collecting data, applying a small processing if required and then delivers the data at a service level. At service layer, BD focuses on storing and performing analysis of large quantity of data collected by the instance.
In the diagram Figure 25, there are three components:

- Collector
- Adapter
- Loader

The collector component exposes an interface that other components can use to send data to the BD manager. In this way only when data is generated and available it is sent to the instance for storage. This mechanism is passive and allows scheduling data delivery according to available resources. The other mechanism is active and requires the collector to know the source of data so that it can regularly collects information from Things that need to be stored.

Once data has been taken by the collector it can be adapted by a component that wraps a number of available adaptations. This component can be not installed on gateway where this processing of collected data is an excessive burden for the available resources.

Collector transfers the data to a Loader component. This component stores data in a local storage area when resources are available. It has the purpose of loading data in to a service layer component that will move it to another stage for long term storage. To do this, it expose interface for active and passive querying of available data. It is separated by the collector because in this way mechanism of retrieving and querying of available data are decoupled from mechanism of collecting data.

Figure 26 shows the components of the service layer. When the loader component has data available it sends it to a data receiver located in the service layer of BETaaS. This component can be on the same gateway or in another one. Moreover if this data receiver is in the same gateway that also is attached to the storage, then it has the storage deliver and manager components active and available. Storage deliver takes data, applies transformations so that data format is homogeneous inside the local file system and then, by using the storage manager interface, performs the necessary operation in order to transfer information on the storage system. The storage manager wraps the utility to control and perform I/O operation on the file system.

An application that wants to perform a task over BETaaS data, need to make use of the application data manager interface that expose the API necessary to get information about available data and task that can be performed by applications. When the applications request an analysis of recent data it sends parameters to application data manager that use the query system to build real time query and run them over the storage to get the results. An application, in order to receive the result of a data analysis performed by a job, sends a request to the application manager that will check on the scheduler the active task and when the results are available will send them to the requesting application.
5.7.2. Package Provided External Interfaces

IF_BD:pushData(data). Used to receive data from another gateway.
IF_BD:pullData(data). Used to actively receive data from another gateway.
IF_IM: execTask(taskId,param1, param2,...) Used by IM to start a task on the BD manager.
IF_IM: dataAvailable(taskID) Used by the BD to inform that a task has completed.
IF_IM: taskDataAvailable(taskID) Used to retrieve data for a terminated task.
IF_IM:getTaskAvailable() Used to retrieve the current task registered under the Big Data manager.

5.7.3. Package Required External Interfaces

IF_BD:pullData(data). Used to get data from another gateway.
IF_IM:removeBD() Used by a BD to remove itself from the instance.
IF_IM:getAvailableBDGateway() Used by BD to get the list of available Big Data enabled gateway.
IF_IM:registerBDService(gwID) Used from BD to register itself as a Big Data provider.
IF_IM:synchronizeData(instanceInfo). Used by BD to get information about the current instance of BETaaS.
IF_IM:registerNewTask(param1,param2..) Used by BD to tell to the IM that a new task is available.
IF_CM: setThingsBDM(thingsList[thingID, data]) BD requires to retrieve for each things available the data it returns.
IF_CM: deleteThingBD(thingsIDsList) Used by CM to delete the things not anymore available in the BETaaS instance.
IF_StorageManager::write(file, data) Exposed by the storage system and used by BD to write data on the file system.

IF_StorageManager::read(file) Exposed by the storage system and used by BD to read data from the file system. Data can be output of a task run on data previously.

IF_BD::runTask(job) Provided by Big Data system and used by BD to run an existing job over all available data.

IF_BD::executeQuery BD use this interface to perform a query over the existing data. It can be provided by the Big Data system or a traditional Data Base system.

5.8. VMs Manager

5.8.1. Component Architecture

In those gateways with enough computing power and hardware compatible with virtualization, it is possible to start small VMs in which some services can be executed in an isolated way and storage can be exploited for Big Data analysis.

The next figure shows the architecture of the component. The VMsAllocator is the one centralizing the interactions in the component. It is the one receiving the request for performing actions on VMs, so it will determine the resources to allocate. It may decide also to use an external cloud or to ask to other gateway to create a VM.

![VM Manager architecture](image)

The HypervisorManager is the one with direct contact with the hypervisor, so it will be the one requesting the hypervisor to perform actions on the VMs (create, stop, start, etc...), according to the requests coming from the VMsAllocator. It will also provide information about the VMs to the VMMonitor. The VMMonitor will retrieve data about the VMs and it will share it with the VMsAllocator and the BDManager (which could extract more information analysing it).

Finally, we have two components for using external clients. On one side, we have the CloudsClient for connecting with external clients (using protocols such as OCCI[5]), while the VMMClients will act as a proxy for contacting with other components (VMMs in other gateways, the SecurityManager and the QoSManager).

5.8.2. Package Provided External Interfaces

- **IF_VMsAllocation::CreateVM** – It is used to create new VMs based on a set of requirements provided. It will be used mainly by the TaaSRM, but it could be used by the BDM as well.

- **IF_VMsAllocation::DeleteVM** – It is used to remove those VMs which are not necessary.
- **IF\_VMsAllocation::ModifyVM** – It is used to modify the status (pause, stop, start…) and characteristics (CPU, memory…) of a VM.

### 5.8.3. Package Required External Interfaces

- **IF\_Hypervisor** – It is used in order to access directly to the hypervisor which manages VMs directly in the infrastructure.
- **IF\_BDM** – it is used in order to push some data to the BDM, monitored from the VMs controlled by the hypervisor.
- **IF\_OCCI** This interface is used to send REST requests, according to the OCCI specification, to external cloud platforms.
- **IF\_ExtVMM** – it is used in order to interact with those VMMs located in other GWs, but belonging to the same instance.

### 5.9. Things Adaptor

#### 5.9.1. Component Architecture

The part of the Adaptation layer which deals with Things and the management, in general, is a module that acts as the bridge between the physical layer and the BETaaS platform as described in chapter 3.

In particular the ThingAdaptor is publishing an interface to the modules requiring access to the Things. The Adaptor’s internal processes and burdens are handled by the AccessThing and ThingsManager component which are responsible for orchestrating the communication whenever required by outside modules through the ThingsAdaptor. AccessThing is managing the basic communication through the ThingsDiscoverManager and ThingsManager is delegating particular requests to specific Things. The latter also provides readings and measurements to the ThingsAdaptor.

The particular package which provides the TA functionality to upper layers and the TaaS in particular is providing an interface to it via **ThingsAdaptor** class. Additionally in order to communicate with the layer below, the Physical Layer, it must implement the required API (i.e. of the CoAP) in the **ThingsDiscoverManager**.

![Figure 28 Things Adaptor architecture](image)

### 5.9.2. Package Provided External Interfaces

- **IF\_TA::getThingsConnected** – It checks that a set of things needed for executing an application or a service are connected and are ready to be used.
- \textit{IF\_TA::getRealTimeAdaptedInformation} – This can be used for retrieving real time data from sensors.
- \textit{IF\_TA::setActuatorsAdaptedValues} – It sets certain values in actuators, so they will perform certain actions.
- \textit{IF\_TA::findThings} – This requests to discover those things available.

5.9.3. Package Required External Interfaces

- \textit{IF\_PhysicalLayer} – It is required for accessing to the physical layer. It will depend on the M2M solution used.
- \textit{IF\_CM} – This component will be providing information to the Context Manager through this interface.
6. Platform Deployment

The first point to take into account is that there is a clear difference between the logical gateways and their physical representation. We can consider set-up-boxes and routers as devices for hosting BETaaS gateways, while a BETaaS gateway, in the end, is an instantiation of the architecture presented in this document. In general, one physical gateway (one device) will host only one logical BETaaS gateway but this does not mean that another possibility is not possible.

In those physical devices with enough resources, it would be possible to run two logical gateways over the same hardware. In this case, the rationale for doing such a deployment would be because one of the gateways controls a limited set of things or because we want to keep two logical gateways, each one joining different BETaaS instances.

There are several ways for deploying the architecture components in real devices. We foresee two main distinctions in the kind of devices which will host gateway components:

- Those with minimum computation requirements
- Those with many computation and storage requirements, compatible with virtualization mechanisms.

In the first case, those components deployed will be the basic ones, in order to enable basic capabilities and the extended capabilities for QoS, security, dependability and reliability. It is also possible to have some Big Data capabilities, but they will be limited due to the hardware constraints.

In the second case, it will be possible to add full Big Data capabilities, together with virtualization capabilities, so VMs can be created and used, enabling the possibility to have isolated computation resources to be used as a way to support Big Data analysis or services execution.

![Figure 29: BETaaS Deployment Diagram](image)

In the case of the diagram above, GW* would be a gateway providing all the basic functionalities, but with hardware limitations. On the other hand, GW1 would be a full capable BETaaS gateway, with enough hardware capability to support also virtualization. This is an example in which, moreover, GW* is the alpha gateway managing the BETaaS instance, which means that it is not necessary that the most capable gateway performs the management role.

As shown in the deployment diagram, both gateways provide three interfaces and require three interfaces from the other one. These interfaces represent:

- **IF_ServiceLayer:** Sometimes it is necessary that components at the service layer, but belonging to different gateways, interact (i.e. for synchronizing the list of services available). This interface is the one used for enabling this kind of interactions at the service layer.

- **IF_TaaSLayer:** As in the case of the service layer, the TaaS layer also requires some interactions between components at the same layer and different gateways. An example would be the synchronization of things services between TaaS Resource Managers.
- **IF_AdaptationLayer**: Again, it may happen that components at the adaptation layer need to interact. It could be the case of Context Managers or even Things Adaptors for interchanging certain information about connected things.

The interactions among components in different gateways will depend on the design and needs for each case. Usually, these interactions will happen between components of the same class (i.e. a TaaS Resource Manager will interact with the TaaS Resource Manager in the remote gateway). These interactions have been already defined for each component.

The following diagram describes the deployment of the module among GWs for the Dependability Manager, one of the subcomponents of the Security Manager. In this case, each GW is monitored by its own Monitoring Module and the recovery policies are planned by a designed GW*. These interactions would happen at the TaaS layer, between two instances of the Dependability Manager, each of them located at different gateways.

![Dependability Deployment Diagram](image.png)

**Figure 30: Dependability Deployment Diagram**
7. Conclusions

According to the BETaaS conceptual model and its defined capabilities (both basic and extended ones), we have identified the main functionalities to be provided by BETaaS instances. We have identified the following groups of functionalities: Things Access Adaptation, Resources Management and Discovery, Security Services, QoS Services, Big Data Management, Context Management, Services and Applications Management and, finally, Instances Management. All these functionalities can be mapped to the capabilities listed for the three conceptual layers covered in BETaaS (Adaptation, TaaS and Service layers).

As we understand these functionalities cover all the needs in BETaaS, a high level architecture has been defined, in order to define the way to implement these functionalities. In order to check that the architecture is valid, we generated some sequence diagrams based on the Home Automation scenario. Some of the steps of the use case were analyzed and we showed how they would be performed in a BETaaS instance with the proposed architecture.

Based on the proposed architecture, detailed designs have been defined for each component. In these designs, there has been a coordinated work with the APIs definition, so not only internal architectures have been defined, but also the interfaces required and provided. This facilitated to have a consistent architecture with the interactions which happen among components and gateways.

The result is an architecture with the enough detail to start the first iteration of the implementation in which we will be able to obtain modular components which can evolve and be improved in each iteration.
8. References

[1] Building the Environment of the Things as a Service (BETaaS) - EU FP7 Project, Deliverable D1.4.1 – TaaS Reference Model

[2] Building the Environment of the Things as a Service (BETaaS) - EU FP7 Project, Deliverable D2.1.1 – Specification of the basic capabilities and content use of the platform

[3] Building the Environment of the Things as a Service (BETaaS) - EU FP7 Project, Deliverable D2.2.1 – Specification of the extended capabilities of the platform

[4] Building the Environment of the Things as a Service (BETaaS) - EU FP7 Project, Deliverable D1.3.1 – Use cases and KPIs.