****

**Enhanced Affective Wellbeing based on Emotion Technologies for adapting IoT spaces**

**Multi-layered Architecture for EmoSpaces Platform**

FINAL

|  |  |
| --- | --- |
| **Consortium:** | **Emospaces Consortium** |
| **Deliverable ID:** | **2.4** |
| **Work package/Task: WP2/T2.4** | | |
| **Responsible partner:** | **ITI** |
| **Contributing partner(s):** | **HUFS and ITI** |
| **Dissemination level:** | **Public** |

🙔 This page is intentionally left blank. 🙔

Index

INDEX

[1. Introduction 8](#_Toc9586157)

[2. Scope and deliverable objectives 8](#_Toc9586158)

[2.1. Scope 8](#_Toc9586159)

[2.2. Deliverable objectives 8](#_Toc9586160)

[3. International consortium architecture 8](#_Toc9586161)

[4. Multi-layered Architecture for EmoSpaces Platform proposed from Korean Consortium 9](#_Toc9586162)

[4.1. Multi-layered Architecture Model 10](#_Toc9586163)

[4.1.1. Architecture Details 10](#_Toc9586164)

[4.2. Service Layer 12](#_Toc9586165)

[4.2.1. Service Level Components 12](#_Toc9586166)

[4.3. Composite Virtual Object (CVO) Layer 13](#_Toc9586167)

[4.3.1. CVO Level Components 13](#_Toc9586168)

[4.4. Virtual Object (VO) Layer 15](#_Toc9586169)

[4.4.1. Virtual Object Level Components 15](#_Toc9586170)

[4.5. Physical Layer 16](#_Toc9586171)

[4.5.1. Communication Interfaces 17](#_Toc9586172)

[4.5.2. VO communication 17](#_Toc9586173)

[4.5.3. Providing VO Sensor Data 20](#_Toc9586174)

[4.5.4. VO Back-End (Interface with RWO) 21](#_Toc9586175)

[5. Multi-layered Architecture for EmoSpaces Platform proposed from Spanish Consortium 23](#_Toc9586176)

[5.1. Functional Architecture 24](#_Toc9586177)

[5.1.1. User processing interface 25](#_Toc9586178)

[5.1.2. Distributed processing layer 27](#_Toc9586179)

[5.1.3. Distributed storage layer 29](#_Toc9586180)

[5.1.4. Streaming layer 31](#_Toc9586181)

[5.2. ITI BDAaaS Architecture 32](#_Toc9586182)

[6. Conclusions 33](#_Toc9586183)

**Document history**

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision** | **Date** | **Content description/Modification** | **Author(s)** |
| V0.1 | 21/06/2018 | Creation | Ilyoung Chong, (HUFS) |
| V1 | 23/06/2018 | Minor Typo Errors Correction | Ilyoung Chong, (HUFS) |
| V2 | 16/04/2019 | Adapting deliverable to the Spanish use cases | Mario Rodrigo Solaz, (ITI) |
| V3 | 02/05/2019 | Additional revision | ITI |
|  |  |  |  |

**Glossary**

|  |  |
| --- | --- |
| CVO | A collection of multiple VOs to abstract a service feature, operation or management function, to enable the mash-up and collaboration. |
| EmoSpaces | Enhanced Affective Wellbeing based on Emotion Technologies for adapting IoT spaces |
| ITEA3 | Information Technology for European Advancement 3 |
| RWO | Real world objects are identifiable through their virtual representation. |
| VO | A virtual representation of a real world object (e.g., sensor, device, task, process and information). |
| WoO | A way to incorporate virtual objects on the World Wide Web and to facilitate the creation of IoT services. |
| BDAaaS | Big Data Analytics as a Service |

# Introduction

In the internet evolution, Internet of Things (IoT) is becoming one of the major step by harnessing billions of sensors and linked devices to acquire data for decision making. With continuous improvements in IoT, protocols connecting the things and state-of-the-art middleware solutions; a new direction is to enable these developments to provide services closer to user preferences and emotions. Emotion aware technologies can leverage IoT based smart space environments to provide more personalized future services. In people’s daily life, understanding the emotion plays a critical role. Emotion awareness could be very helpful in various domains. For instance, to support depressive people by providing coaching and assistance to give them a way to overcome their hopelessness or to help elderly by providing smart service provisioning in their daily living environment as close to their emotional needs and preferences as possible.

To support emotion-based services, IoT architecture should incorporate the functional components that help collect, fuse, reason and analyze the data from multiple sources. It should also provide means to handle and manipulate several types of data and support context-aware, personalized service provisioning. In sensing emotion in spaces, these required features provide multiple layered structure to support efficient processing of sensed and collected data to represent emotion well.

# Scope and deliverable objectives

## Scope

The objective of this deliverable is to state the multi-layered architecture for EmoSpaces platform. This architecture will describe the essential components required to support EmoSpaces platform.

The consortium will demonstrate the multi-layered architecture functional components to deploy Internet of things (IoT) services in the World Wide Web environment.

## Deliverable objectives

This deliverable aims at describing the multi-layered architecture and its components, describing how the EmoSpaces platform will be used to deploy and support the EmoServices.

# International consortium architecture

Each one of the consortia involved in the EmoSpaces project started from its own architectural proposal. These proposals were influenced by the needs and particularities of the use cases to be supported. In this section, a general representation containing all the architectures contained in the project is shown (Figure 1). In this diagram, it is observed how the different decisions taken by each consortium come together in a common architecture.

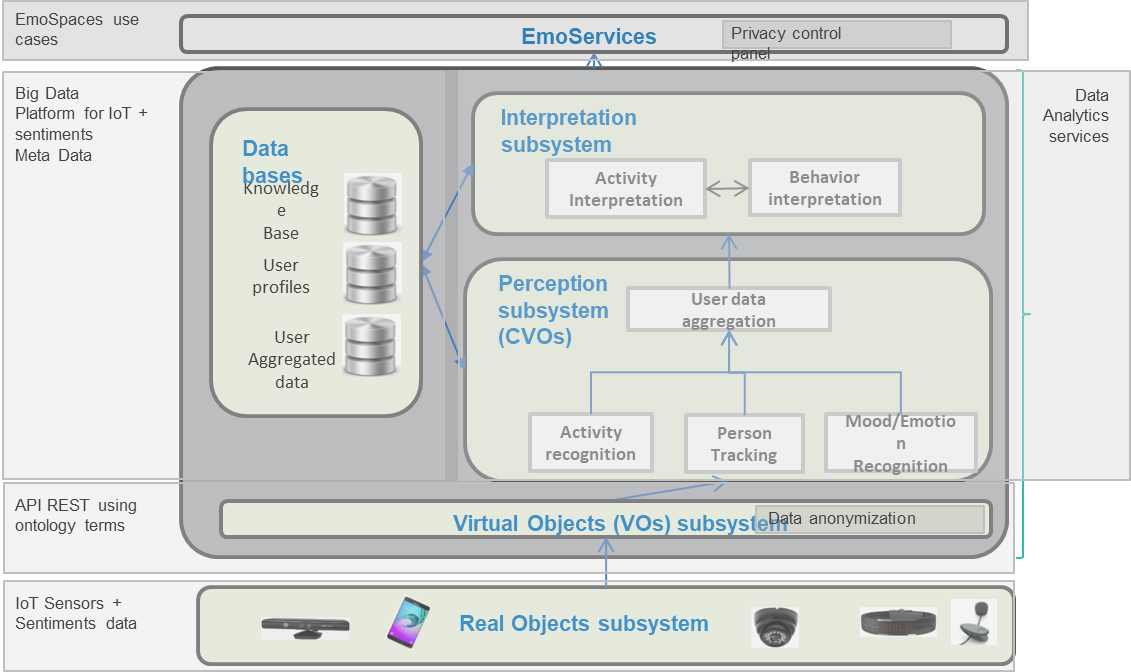


Figure 1: International consortium architecture

Figure 1 shows the different layers that compose the global architecture. At the bottom, we have the data sources, i.e. IoT sensors and textual data. On the opposite side of the graph, at the top, we have the use cases of EmoSpaces. These use cases will use and exploit the information contained at the platform. Finally, in the central part of the picture, we have both the analytics and the storage functionalities offered by the platform.

Two items should be highlighted: “Privacy control panel”, located at the level of emoservices and “Data anonymization”, located at the Virtual Objects layer. In this privacy control panel, users can access and modify the consent for data exploitation at any time. The data anonymization module is where raw data is transformed to remove all personal information.

Next sections will present the two approaches for the global architecture proposed at EmoSpaces project.

# Multi-layered Architecture for EmoSpaces Platform proposed from Korean Consortium

## Multi-layered Architecture Model

The Multi-layered Architecture for EmoSpaces Platform from bottom to top is composed of a physical layer to represent real world objects (including sensors, actuators), Virtual level which is divided into two sub-levels (i.e., VO sub-level level, CVO sub-level) and the service level which delivers an interface to the services provided by the virtual level. The application layer represents applications that will consume the functionality of the bottom layers to provide emotion based services to the end users.

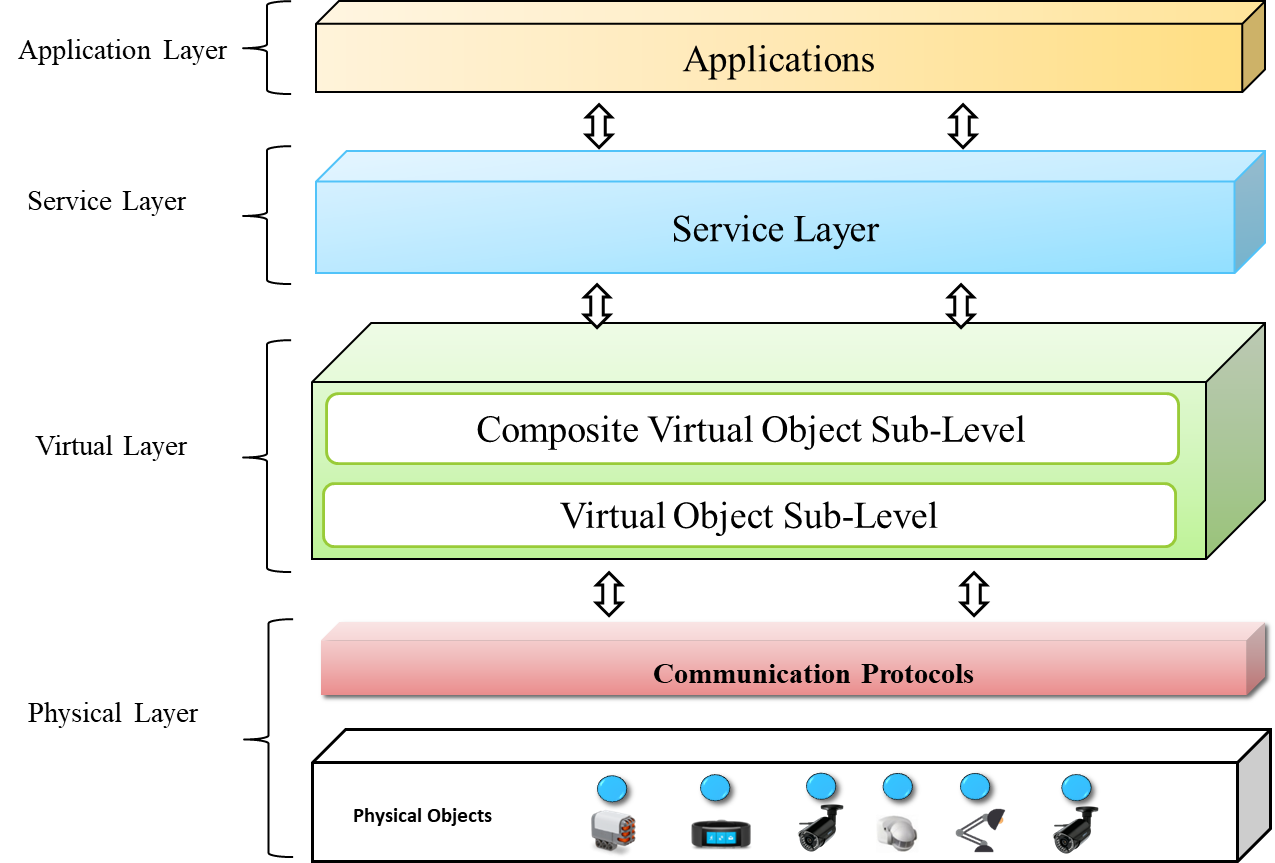


Figure 2: Multi-layered Architecture Model

### Architecture Details

The architecture and the associated components are shown in the figure 2 and their main functions are described as follows:

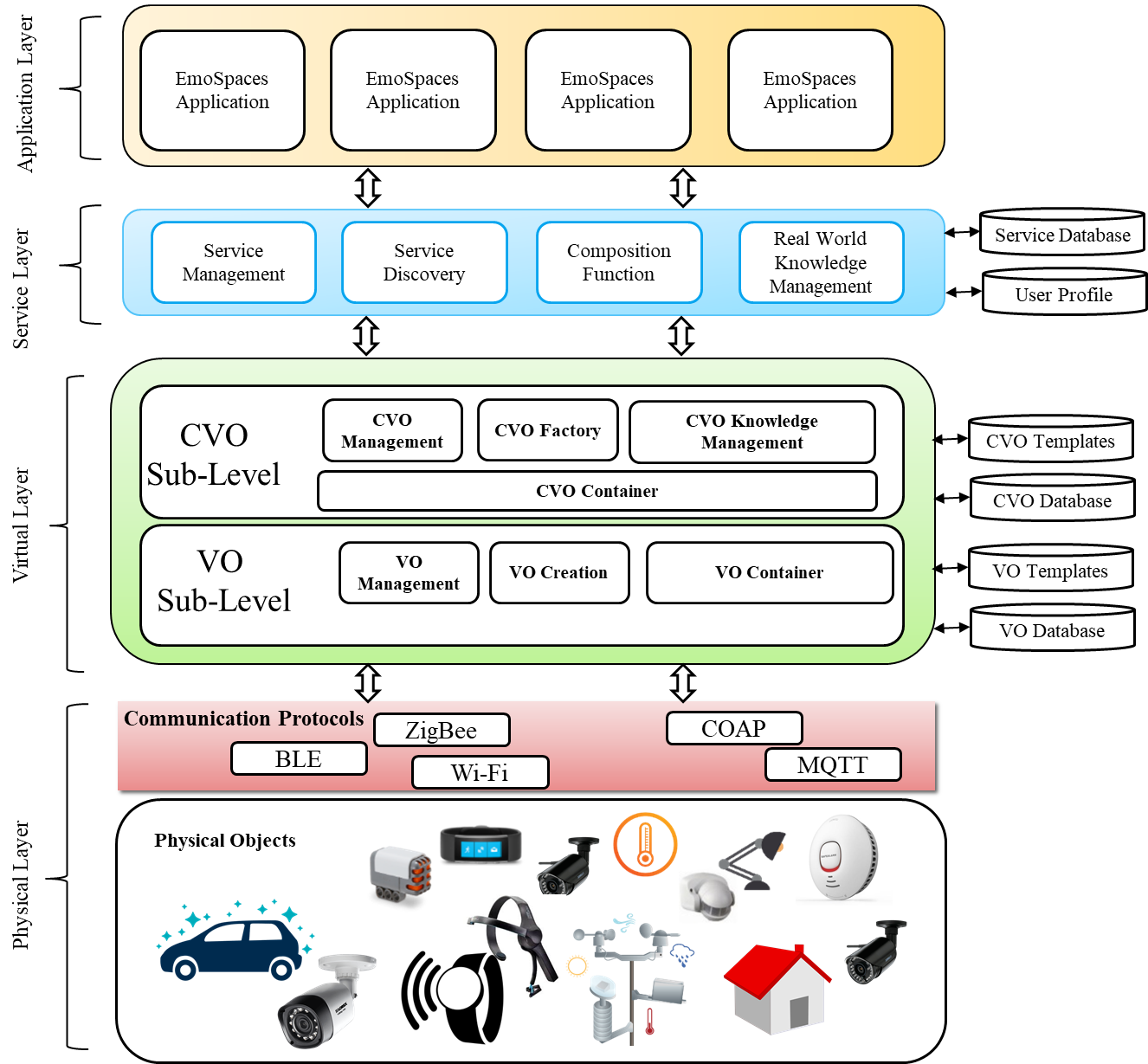


Figure 3: Functional Components in the Architecture

Multi-layered architecture is distributed into physical, virtual, service and application levels. Whereas Virtual Level is further partitioned into Composite Virtual Object Sub Level and Virtual Object Level Sublevel.

Service layer constitutes the necessary components to support applications. Service layer is supported with databases to semantically represent service related data.

Composite Virtual Object provides functions to support composition of virtual objects. CVO level make use of CVO repositories to store and manage CVO data and register entries in the CVO data base.

Virtual Object Layer provides necessary component to digital represent physical objects in the form of virtual objects. VOs are semantically represented in repositories at this level.

The physical layer in this architecture comprises the necessary protocols and interfaces to communicate with the real world objects (i.e. sensors and actuators).

## Service Layer

### Service Level Components

In Top down fashion, service level is one of the first interaction points for applications. Service level incorporates several functions such as service management, configuration and execution function and real world knowledge management function. The service level also provides the composition function which enables the orchestration of services to achieve the service request objectives.

#### Service management function

At the core of service level, the service management function consists of tasks from request inception to service configuration, registration, template management and controlling service execution.

Further components that support the tasks of service management are the following. The Service Request Evaluator component **i**ncludes mechanisms to parse and evaluate the service request so that most suitable services can be selected. Service Controller component involves matching service requests to service instances, and handle service management function. The Configuration Management function includes functions to configure software components settings based on service requirements. Service Template function is the part of the service function that includes mechanisms to create, manage and update the service templates. The Service Registration Manager handles service registry, and provides interfaces to external stakeholders to add and modify service templates.

The service management also handles service level databases. It supports to store and retrieve RDF about description of available services. At service level, semantic data handling services include the mechanisms to handle Sparql end point towards these databases. Service databases support creation of semantically enriched description of services, the interaction with stored data, and communication with other entities.

#### Service Discovery Function

The service discovery function analyses the service request based on user information and supports the discovery of services using query management functions. It also maintains the cache logs to efficiently handle similar requests. Service discovery function queries the services DB to access the current services. It also uses service history, semantic matchmaking and service constraints in the discovery process.

#### Service Composition Function

Composition function facilitates the creation of composite services to satisfy user requests. It includes, service selection, workflow management, binding function to achieve composition of services. The Service Selector components is responsible for selecting appropriate services as defined by Workflow Manager. The Workflow Manager is the core component in the composition function, which forms the composition flow among various services, based on their inputs and outputs. The Binder associates services in a composition graph. The Service Coordination function identifies the dependency of each service in the composition in order to avoid conflicts, managing them in case one occurs during service execution.

#### Real World Knowledge Management

The Real World Knowledge (RWK) Management constitutes the functions to manage real world information associated with the objects. It represents the current and past situations and user data in the form of real world knowledge. This function also manages a RWK database which includes a knowledge and situational facts related to real world objects. The User Situation Model is handled here to get user situation information. It also includes Situation Managements function, such as detection, classification and recognition.

#### Service Execution Environment

The service execution environment is a containerized execution environment for service execution. It constitutes other function such as, Service Life Cycle Manager, to handle service switching among different states from active to release. It supports service monitoring, control and management. The Life cycle manager supports services which drive through a number of different states during its life cycle, such as service creation, execution, suspension, and termination. Furthermore, Service Cache is also maintained to store reference to the instances of recently used services

#### User management function

The user management function includes several components such as Service User Registration function to support user registration for the services. The Service membership management provides the functions such as the creation, maintenance and release of service user membership profiles. It also allows joining and leaving of users for particular services. The service membership management interacts with service management functions to handle membership for user groups.

## Composite Virtual Object (CVO) Layer

### CVO Level Components

In a layered architecture, a composite virtual object (CVO) contains at least one or more virtual objects (VOs) to achieve specified tasks in the Emospaces service provisioning environment. The CVO may also contain or access the data of other relevant CVOs. In this architecture, the mashup or data aggregation of multiple VOs and CVOs is an essential part of the virtual layer. The CVO may also contain some machine learning algorithms for better aggregation and classification of the data received from the multiple VOs. CVOs are self-managed, and self-configurable components which use artificial agent mechanisms to enable the reusing of existing VOs and CVOs for the provisioning of services to the service layer.

The CVO sub-level in the layered architecture contains many necessary relevant functions such as CVO management, CVO factory, CVO container, and CVO knowledge management functions. The description of each component of CVO sub-level is explained in the following sections.

#### Management of CVOs

The CVO management contains the functions to manage the lifecycle of CVOs. The lifecycle includes the selections of CVOs for the execution of service features, the creations of CVOs, the right management of CVOs, updates of CVOs, and maintenance of CVOs. The CVO management functional module also works as an initial landing field from the service level request, it coordinates the process and tasks among the various sub-functions of the CVO management module.

#### CVO Factory

The CVO factory is an abstract module that provides a factory component for the creation of CVO instances. This module also includes the logic for the functional optimal matching of service execution request over the existing CVO level infrastructure. The CVO factory also contains the correlations of functionality aiming the generation optimization and long-term use of CVO and orchestration. The CVO factory also contains the generic templates of CVOs from which other CVOs will be created when the CVO creation request is triggered. A generic CVO template is a general structure that can be enhanced or extended to any other CVO formation.

#### Knowledge Management of CVOs

The CVO knowledge management module manages the data that have been used in a situation or service request. That knowledge will be reused in other service requests for detection such situation in the real world. This module also manages the history of CVOs which have been used in the similar situation. The history of CVOs usage and relevant metadata is stored in the knowledge database.

#### CVO container

The CVO container provides the execution environment for executing the CVO. The container bundled with necessary software which is used during the execution of CVO. The CVO execution process considers all standard aspects which guarantee the availability of the CVO after it has been created until it is destroyed. The execution process concerns mostly runtime mechanisms in order to keep the CVO active so that it is available in order to perform functions fulfilling Service Level service requests or to publish its relevant state changes appropriately. This includes appropriate CVO containers and other IT infrastructures to execute the necessary IT realizations. Those infrastructures must offer a stable environment for the CVO execution independent of external (VO-related) failures or the mobility of the CVO and its containing CVOs and VOs. During execution, CVOs have to access other CVOs and especially VOs in order to realize their functionality.

## Virtual Object (VO) Layer

### Virtual Object Level Components

Virtual Object is any entity that represents a Real World Object such as Sensor, Actuator, process, task, data, and information. Virtual Objects can be seen as providing a very basic set of functionalities representing the actual functions of Real World Objects. Real World Objects are physical objects in an IoT environment that can get connected to the multi layered platform when they are capable to deliver required Information to the System and meet the condition of the physical object itself. Real world objects are identifiable through their virtual representation. The VO sub-level in the layered architecture contains many useful functions such as VO management, VO creation, and VO container.

#### VO Management

VO management unit is responsible for the lifecycle management, resource optimization and dynamic link association management functionalities. VO management function includes access controller which maintain the list of CVOs, who has the permission to access the parameters in the VO instances within its management domain. When this block receives an access request, it forwards the request to the access conflict resolver. It provides the access right to the requesting CVO after getting a green signal from the conflict resolver. When a new VO is installed, the VO management unit fetches the proper VO template from the VO factory.

#### VO Creation

The VO creation module contains the functionality for the creation of VOs. The VO is created based on one or more key functionalities. The VO creation has two parts: VO template creation, and VO instantiation. VO template is created so that similar new VOs can be described easily by reusing the previous template. A template is created based on the key functionality required by the description and involved RWO(s). A manufacturer of an RWO can create VO templates for their products. Alternatively, the system administrator can also create VO templates. The VO instantiation is the process that activates the VO functionality and VO designated tasks.

#### VO Container

VO Container is the run-time process of a VO including both VO front-end modules for interacting with the upper levels, and back-end modules for interacting with the ICT objects. The VO container has front-end and back-end part. The front-end communicates to the upper levels with common web technologies. Thus, the VO Container essentially acts as a gateway between the Internet and constrained battery operated networks. The back-end part of the VO Container communicates with either a gateway or a sensor/actuator allowing generation of the awareness required to estimate the status of the association between the real world objects.

#### VO Registration Mechanism:

The registration mechanisms take over the registration of new VOs in the VO Registry. More specifically, in order to enhance the registration process, a set of VO description templates can be used so as to describe the VOs in terms of parameters that are defined in the VO Information Model. These descriptions can be created in the XML or JSON format. The XML/JSON data can be sent to VO Registry and the VO registration mechanisms. The RDF data are stored in the VO registry.

#### VO Discovery Mechanism:

The VO discovery mechanisms enable the search and discovery of the available VOs in the system. An external entity can use the VO Registry Client to perform discovery requests to the VO Registry, using certain search constraints. It also considers the access rights regarding the VO and the VO Registry client that performs the discovery request.

#### VO Access Control Mechanism:

The Access Control mechanisms take over the control of the accessibility on VO description data by external entities, based on specific Access Rights. These Access Rights define whether a function or parameter can be read or manipulated by specific user or a specific user role. Consequently, it is able to provide individually assign read/write Access Rights to every VO function or VO parameter. Thus, a VO Registry client that is associated with a specific user role and has specific access rights, each time when it tries to modify the VO data, is checked by the access control mechanisms to be authorized for the modification process.

## Physical Layer

The main task of the physical layer (shown in the figure below) is to perceive the physical properties of things around us that are part of the IoT. This process of perception is based on several sensing technologies (e.g. RFID, WSN, GPS, NFC, etc.). In addition, this layer is in charge of converting the information to digital signals, and vice versa. This layer supports the functions to provide sensed data to upper layers such as VO sub level.

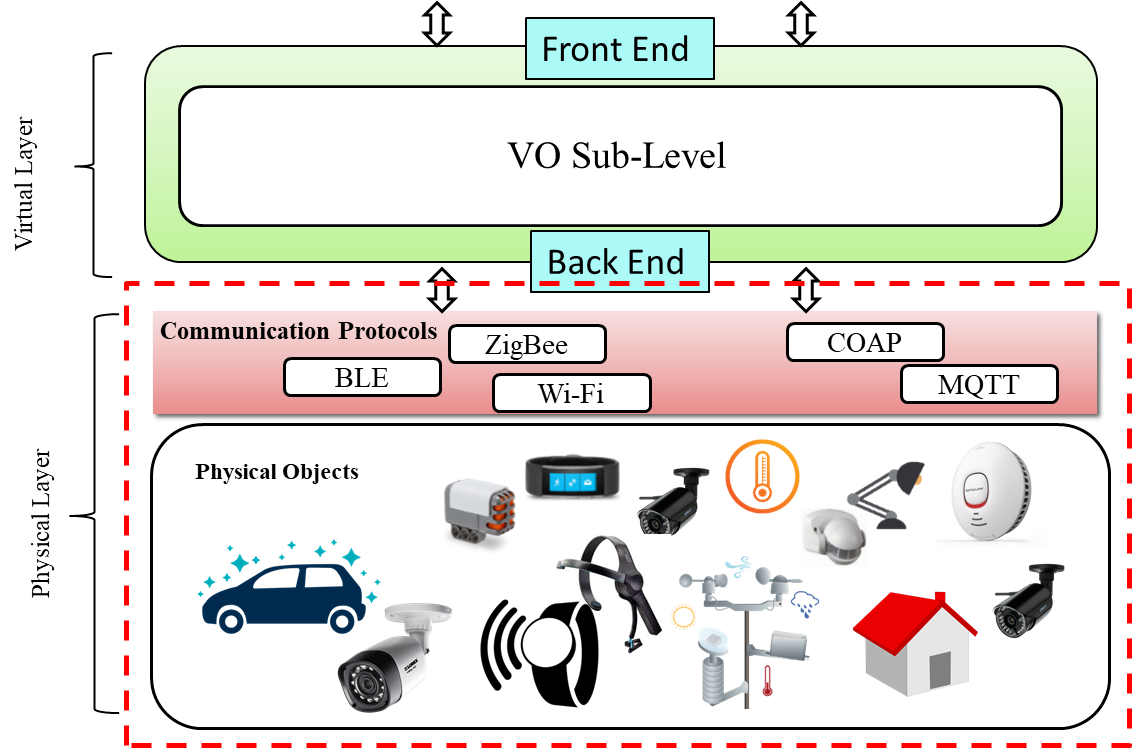


Figure 4. Physical Layer

### Communication Interfaces

VO Sub-Level interfaces are necessary for inter communication of functional blocks. VOs communicate with digital and physical objects via the south bound interface of the VO Backends. These components are drivers which can communicate with sensors and actuators. VO level has to be able to cope with the common communication patterns present on these lower level interfaces.

### VO communication

The communication between VO and CVO can be elaborated in terms of identifying which communication patterns, communication interfaces and implementation options can be used. The Communication Patterns define how data are exchanged, which communication model will be used (broker mediated, publish subscribe, synchronous RPC style), the technologies available to support these models and the way they will be used within the system. Communication Interfaces define which interfaces a VO will have to expose and for which purpose (e.g. for VO control, for VO data access) and their details (parameters, etc.) Implementation Options take into account the needed patterns and interfaces specified at the previous points, the available options (protocols, data encoding, etc.) for implementing such patterns and interfaces.

A Communication Pattern is the messages exchange that takes place when two parties exchange data. The communication patterns for CVO/VO interaction will be described independently from the underlying wire protocol which will be used, to take into consideration different communication protocols, such as a RPC-style mechanism like SOAP/REST or a broker‐based communication paradigm like MQTT. Different options for the communication implementation can be described.

The Synchronous RPC is the simplest communication pattern which can be applied for VO communication. It allows the synchronous message exchange between CVO and VO. In the Synchronous RPC communication pattern, the CVO typically blocks waiting for the inbound message from the VO.

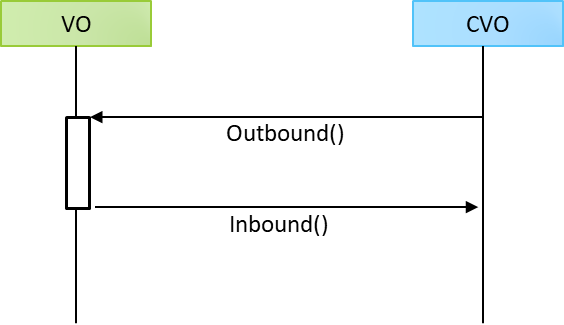


Figure 5. Synchronous RPC Communication

In case of Asynchronous RPC pattern, the CVO will not be blocked waiting for the inbound message, but it will instead provide the VO a communication endpoint for sending back the inbound message, when available, in response to the outbound message.

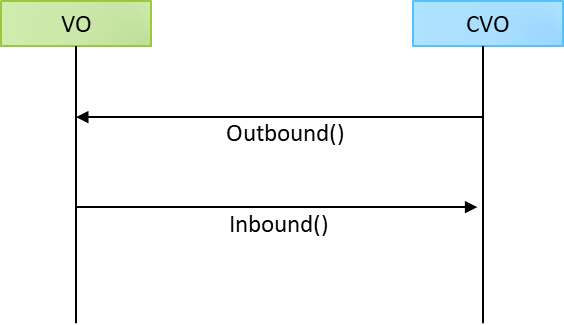


Figure 6. Asynchronous RPC Communication

The Publish - Subscribe communication pattern is again an asynchronous pattern where the CVO subscribes with an outbound message for receiving inbound messages published from the VO, and the CVO will eventually unsubscribe when messages from the VO need not to be received. The Subscribe outbound message will likely contain the type of event that the CVO needs to receive, the frequency and the communication endpoint where the CVO will listen for inbound messages.

In this pattern the publish-subscribe pattern implementation is entire responsible of the VO. From this point of view, this pattern provides a simple choice for implementing a publish‐subscribe pattern, while a more sophisticated behavior can be achieved relying on broker‐mediated architectural choices.

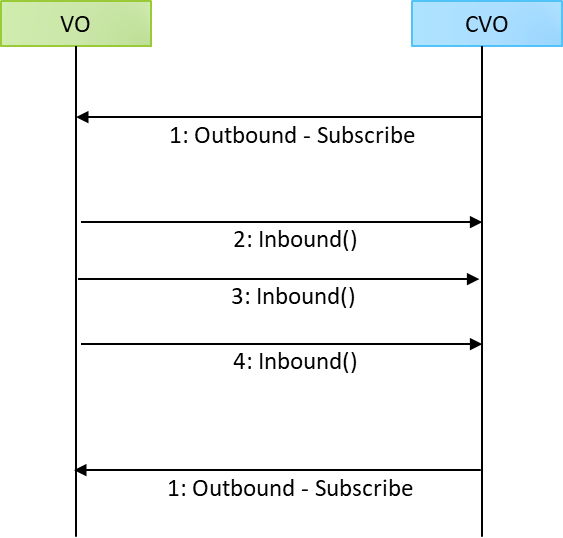


Figure 7. Publish Subscribe Communication

The **Broker Mediated Inbound Messages** pattern defines a broker-mediated message exchange for communicating events from the VO to the CVO (inbound messages). With this pattern, the publish-subscribe mechanism implementation is a responsibility of the broker, which typically manages a set of topics for publishing and subscribing events. The VO will publish events to a specific VO topic, while the CVO will have to subscribe to the same topic for receiving the inbound messages.

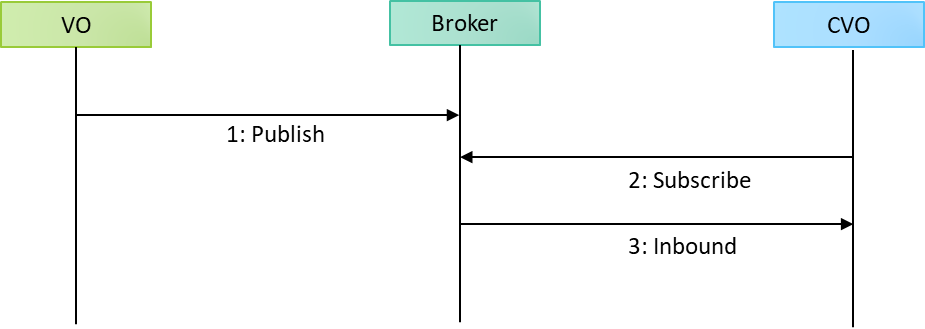


Figure 8. Broker mediated Inbound messages

The Broker**-Mediated Outbound Messages** pattern defines a broker‐mediated message exchange for outbound messages. In this case, the VO will subscribe to the VO specific topic, while the CVO will have to publish the outbound message to the same topic to send the outbound message to the VO. In the case of correlated outbound/inbound messages (Broker Mediated RPC Pattern) the messages will have to hold a specific ID.

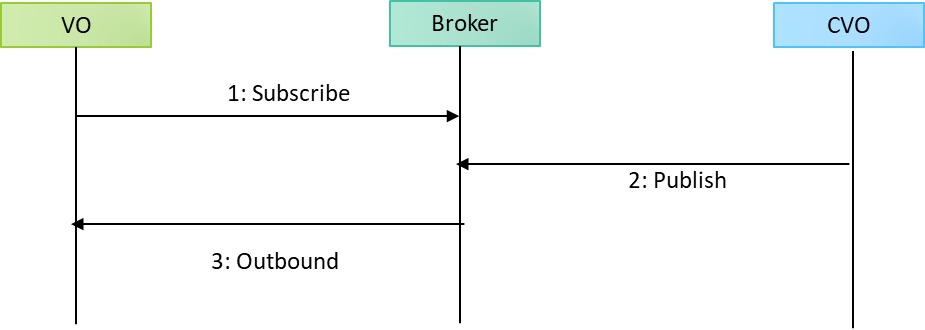


Figure 9. Broker mediated outbound messages

### Providing VO Sensor Data

VO sensor data are related to the measurement of values for the sensor VOs. The set of measures will be sent from the VO to the CVO using different ways according to the specific scenario or sensor technology. All the patterns will be suitable to be applied. The following figure gives an example of the specific interface (content of the message) for sensor data to be communicated from the VO to the CVO.

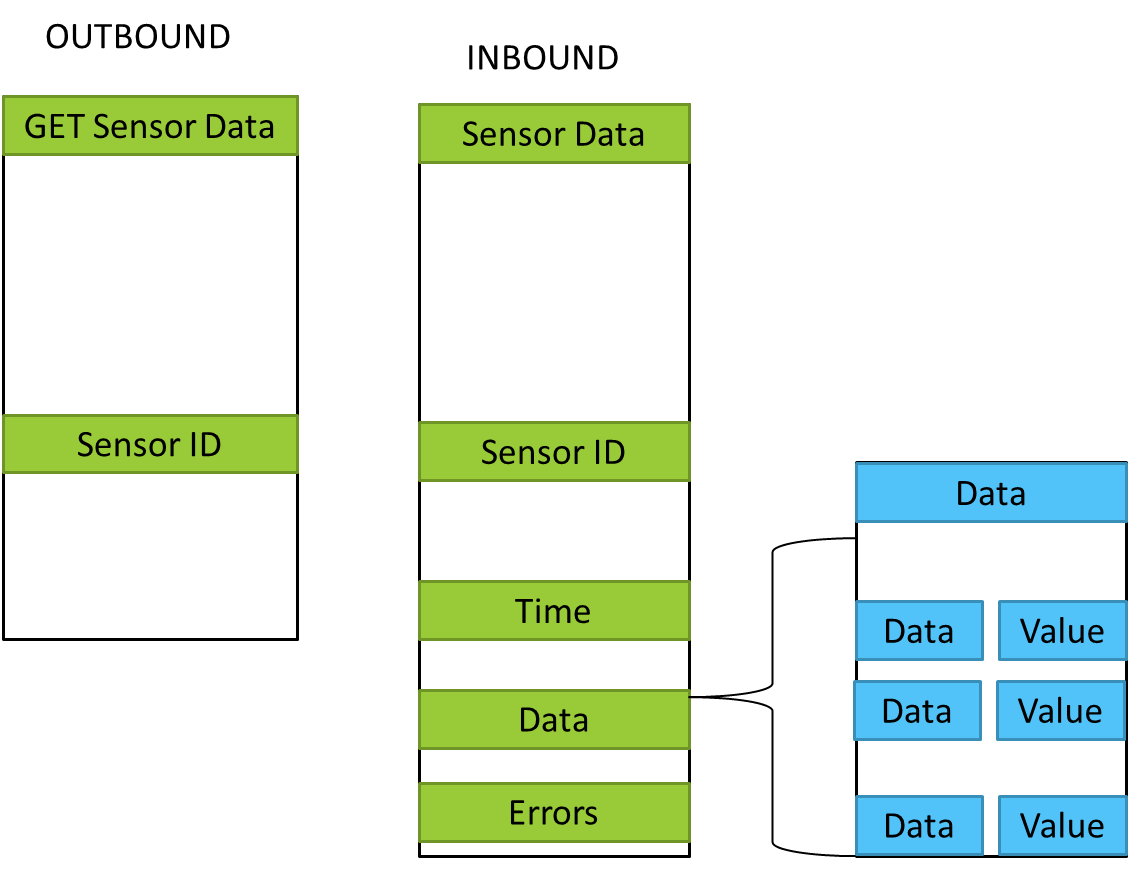


Figure 10. Sensor data from VO and CVO Interface

### VO Back-End (Interface with RWO)

Virtual Objects (VOs) are representations of RWO in the digital world so that any application can access the RWO or its functionality without knowing about the means of physically reaching and retrieving information from it. The term interfacing here means two aspects: (i) interfacing with the RWO and, (ii) interfacing with the higher levels (CVO) and/or other VOs (VOs of other devices) to provide some service. In the above first category, there could be again at least two ways of interfacing: (a) the VO could be a mere abstraction of RWO, i.e. it does not influence the RW and it only represents the real world – for example, Temperature of a room has no physical connection/interface except a transducer catching some electrical signal dependent onto the current temperature; and, (b) the VO eventually influence the RWO by using a direct signaling interface capability with the RWO – for example, an actuator controller where the VO has the ‘bit’ to control the actuator physically. The interface between VO and RWO need to be generic enough so that it can be compatible with any device (RWO) driver. This compatibility can ensure that the action on the VO can be translated by the device driver to do the work physically.

Since a VO is an abstraction of RWO, it helps the application developer to build their application very easily. RWOs need to be reached to carry out a physical work. A VO might reside in a different, distant device (server) than a RWO. In that case, a physical communication between the RWO and the corresponding VO is required. As different RWOs use different physical communication medium, the heterogeneity makes the VO-RWO communication a very tough job. A VO should be able to indicate how a physical communication can be achieved with the corresponding RWO (here RWO refers to an object which can be communicated (queried/controlled) remotely). If a VO represents a Non-ICT object, then the communication indication needs to be made to an ICT device which is monitoring the required functionality in the Non-ICT object. The information a RWO can provide is stored at the corresponding VO using the VO-RWO communication. The digital world gets the information by querying the VO. So the information stored at a VO needs to be accurate and up-to-date/timely. If particular information is not available at the VO, after receiving a query the VO-RWO communication needs to be triggered to get the proper information from the RWO. The VO should know when to trigger a physical communication with the corresponding RWO. Even if there is no query, the VO information needs to be updated periodically (and also the physical communication link to the RWO). In case of performing a function, a request at the VO needs to be translated and communicated for the RWO to perform the function physically – for example, switching ON/OFF. There is also a requirement for VO-RWO communication to update the resource availability at the RWO, especially for those resources that can affect further decision making.

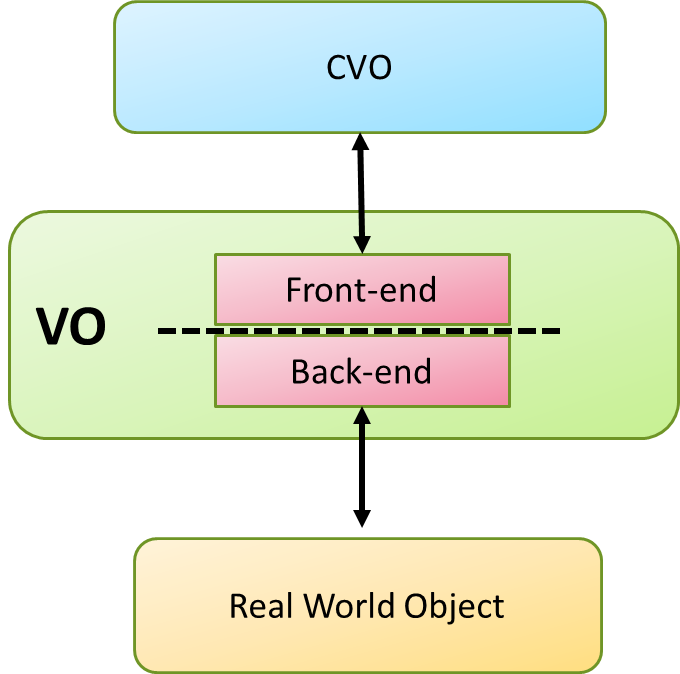


Figure 11. North and South Interfaces

The VO Back-End Software REST Module can be used for wireless sensor networks that handle functionalities and access of ICT objects connected through standardized and proprietary wireless sensor actor networks (WSAN) like ZigBee, LightweightMesh (LWMesh) and ZIGPOS eeRTLS systems.

# Multi-layered Architecture for EmoSpaces Platform proposed from Spanish Consortium

From the Instituto Tecnológico de Informática (ITI), a Big Data platform for the capture, storage, analysis and exploitation of information is made available to the members of the EmoSpaces project. This information can be captured by a multitude of sensors with very diverse capacities, which provide data as varied as the sources that originate them: audiovisual content, wearables and other biometric sensors, IoT sensors and texts. The proposed platform is not only capable of capturing and storing information, but also has the necessary tools to merge it effectively. Together with the platform, or rather integrated in it, ITI provides a set of techniques and algorithms that allow and facilitate the work with the previously stored data. The functions of these tools include, for example, merging data or detecting emotions. In this way, the analyses carried out using ITI's Big Data platform make possible to monitor and understand the behavior and affective state of users, clients and consumers, contained in each of the use cases implemented by the partners.

The main objective of ITI's Big Data Analytics platform is to provide the partners with a BDAaaaS (Big Data Analytics as a Service) solution in the cloud. This platform allows the collection and integration of data from intelligent sensors, as well as the efficient management of large amounts of information, making use of Big Data technologies.

This platform (Figure 13), has been outlined from two different solutions, the first of which is commonly known as "Platform as a Service" (PaaS). The solution proposed by ITI is an elastic platform, which allows the transparent management of resources. And above this PaaS, a final solution called BDAaaS is developed, which contains the necessary tools to support the different use cases implemented by the members of the consortium.

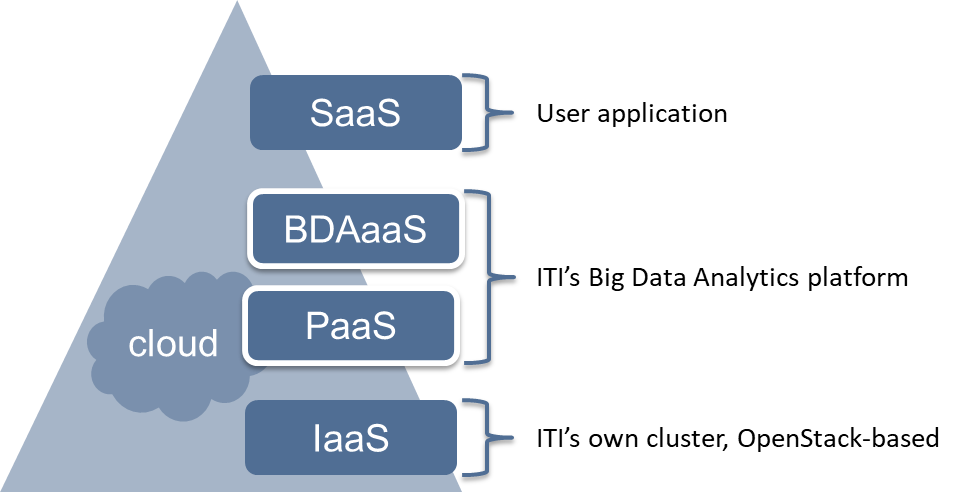


Figure 13: Infrastructure for ITI Big Data Analytics: cloud-based model

ITI's PaaS is designed to easily create elastic services with automated lifecycle management. The goal is to obtain elastic services that adapt to the different operating conditions of the environment (load changes, breakdowns, upgrades, etc.) with minimal effort and expenses on the part of the service provider.

In the PaaS model of the ITI, a Service is a long-term computation, obtained as a result of the execution of a set of software components previously deployed in suitable computer environments, offering functionality accessible through some communication channels, and providing guarantees on how these functions are provided.

These services can be scaled automatically by monitoring the execution of the deployed services, deciding when to increase the number of replicas for each agent, or the resources assigned to each agent instance. The scalable service design adapts the load using a horizontal scalability technique.

ITI's PaaS provides efficient virtualization that deploys agent instances within lightweight Linux containers, providing the right amount of resources needed for agent instances and avoiding costly overloads. The ITI PaaS needs the support of an IaaS below it, and for this, the ITI design allows to use different IaaS providers, or even be run on the ITI cluster itself. Examples of supported IaaS providers are Amazon's AWS, as well as many of the other existing OpenStack-based offerings on the market.

On the other hand, the ITI BDAaaS, deployed on top of the ITI PaaS, aims to allow the partners to focus on data analysis and processing, without being buried by the details of configuration, services, adaptability, deployment and making transparent the selection of the underlying IaaS. The ITI BDAaaS provides an ecosystem of services to address different scenarios such as Predictive Analysis and Exploratory Data Analysis (EDA) based on batch processing (Batch) or real-time (Stream).

## Functional Architecture

The ITI platform for typical Big Data Analytics scenarios can be represented by a set of abstract layers: input data, streaming data acquisition, distributed processing, distributed storage, and output data. Figure 14 shows the functional architecture of the ITI BDA platform and the relationship between the various layers that make it up.

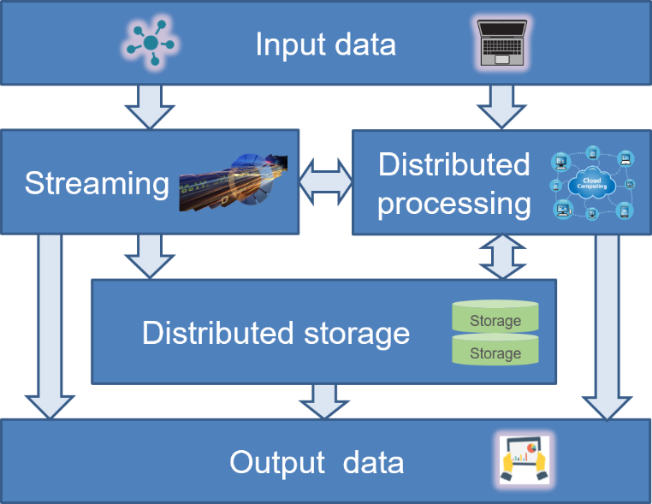


Figure 14: Infrastructure for ITI Big Data Analytics: functional architecture

The *input data* represents the data entered in the platform. These can be obtained from a distributed set of sensors or through a user processing interface. In the latter case, additionally, data scientists can select analytical algorithms already provided with the platform or implement their own ones.

The *streaming layer* manages the constant flow of sensor data, enabling both real-time processing through thedistributed processing phase, and persistent storage in the distributedstorage for further processing.

The results of the analyses performed at the *distributed processing* layer can be presented at the user processing interface (*output data*), stored in the distributed storage or transmitted in streaming (*output data*) through the streaming layer.

Information in *distributed storage* can be retrieved for batch data analysis, can be used from the user processing interface, or can be accessed directly from external entities through an API REST (*output data*).

The technologies required in each of these layers are determined by the data analysis techniques necessary to solve specific problems, with different characteristics and types of incoming data, the amount of data to be processed, the location from which the information is generated, etc. All these factors open up a wide range of possibilities regarding the most suitable technologies to be used and their multiple configuration options depending on the scenario.

The following sections present more detail on the layers of ITI's BDA infrastructure, including its software components and the evaluation of a set of technologies, tested in tool selection.

### User processing interface

Data analysts need some kind of interface to design and code how data will be analyzed and to visualize the results obtained from distributed processing. In data analysis, the most suitable tool for coding and visualization is not a pure code editor, nor an IDE for SQL queries, nor even diagrams for data manipulation (also known as workflows or jobs). It requires a mix of all of them, and that is what notebook platforms can offer.

A *notebook* is an interactive report (usually based on web tools) that it contains:

* The code in one of the supported formats (Python, SQL, R, etc.).
* The result of the code that was executed previously.
* The visualization of the results (graphs, diagrams).
* Anything else such as HTML content, Markdown, plain text, images, etc.

Two of the most popular applications among analysts are Jupyter and Apache Zeppelin. Jupyter is a widely known and widespread software that has been used for a long time in giants such as Google or NASA. Jupyter was created in 2012, is an evolution of IPython Notebook. It is open source, has a large community and a lot of additional software and integrations. Apache Zeppelin was created by the Apache Foundation in 2013, it is also open source, but its community is still smaller than the Jupyter community.

For these reasons, for the ITI platform, Jupyter has been chosen as the supported notebook. Jupyter is a web application that allows to create documents with live code, equations, visualizations and descriptions using Markdown. Jupyter notebooks support more than 40 cores for different languages (e.g. Python, Ruby, Julia, R, etc.) and view results using ggplot2, matplotlib and others. Jupyter also provides an interactive shell for operating system commands and a Qt console for shell-based interactive analysis.

As a complement to the functionality offered by Jupyter, the ITI platform includes another visualization tool highly related to text analysis. Kibana is the default tool for visual exploration and real-time analysis of indexed data in ElasticSearch. It is an open source tool integrated in the ELK technology stack (ElasticSearch, Logstash and Kibana). Kibana allows to design dashboards based on the components such as histograms, line charts, pie charts, heat maps and many more. Using these elements, the user can evaluate the results obtained from data collection and processing using ElasticSearch.

#### Jupyter Component

This notebook has been integrated into the ITI infrastructure as a software component in the ITI PaaS. This component is a reduced version of Jupyter Notebook and has Python language support. It contains the most popular libraries, including machine learning libraries (e.g. Scikit-learn) and numerical calculation (e.g. Numpy). The results can be displayed via the matplotlib library.

This component is composed of:

* Legacy code with the binary Jupyter and its dependencies.
* Jupyter Compoment Class, a class that follows the architectural patterns of the ITI PaaS.
* The manifests necessary for the ITI PaaS, both service and deployment.
* Communication channels for interacting with the ITI PaaS and for linking to other components.

#### Kibana Component

Kibana has been integrated into the ITI infrastructure as a software component in the ITI PaaS. This component is a viewer of the results obtained after applying searches and analyses on the indexed data in ElasticSearch.

This component is composed of:

* Legacy code with the binary of Kibana and its dependencies.
* Kibana Compoment Class, a class that follows the architectural patterns of the PaaS of ITI.
* The manifests necessary for the ITI PaaS, both service and deployment.
* Communication channels for interacting with the ITI PaaS and for linking to other components.

### Distributed processing layer

The principles of distributed computing, also known today as cloud computing, are the key to Big Data technologies and their analytics. Mechanisms related to data storage, access and transfer, visualization and predictive modeling through distributed processing, distributed analysis, on multiple machines, are the key elements that make possible the analysis of large volumes of data within the stipulated costs and times, in a practical way both for human and machine consumption.

An analysis was made of the set of processing technologies that could be incorporated in this layer. Apache Storm was the first considered option. However, this module was discarded due to lack of support from the community that has suffered. The community has moved towards other more popular technologies such as Apache Spark or Apache Flink. Another considered option was WSO2CEP, but it might be inappropriate for some Big Data Analytics scenarios and has a reduced support of programming languages.

Finally, Apache Spark was chosen as the distributed processing engine for the ITI BDA platform. Apache Spark is an open source distributed processing framework that allows users to run large-scale data analysis applications on cluster systems. Originally developed at the University of California, Berkeley's AMPLab, Spark's code base was subsequently donated to the Apache Software Foundation, which has maintained it ever since. In addition, Spark has a large user community and Jupyter Notebook can be easily connected to this component.

Spark is an in-memory data processing engine with well-designed development APIs that enable data workers to process data efficiently, whether in batch or streaming workloads, machine learning tasks, or SQL data loads that require fast iterative access to data sets. Loading large distributed datasets in memory allows Spark to achieve performance improvements. It keeps the original copy intact when data is stored in memory. When an action is called, Spark distributes the data in the node's memory (if it doesn't fit, it will be dumped on disk), performs the calculations and transfers the results back to the controller. Spark performs better at processing large volumes of data than other MapReduce-based technologies such as Hadoop and Storm.

At a high level, Spark uses a master/slave architecture. Basically, it is designed to be used with a variety of programming languages and in a variety of architectures. There is a driver that connects to a single coordinator, called *master*, who manages the *data workers* in which the executors run.

#### Spark Master Component

Spark Master has been integrated into the ITI infrastructure as a software component based on the ITI PaaS. Spark Master is the main component in a distributed process with Spark. This component configures all the necessary elements for the interconnection with the workers, as well as with the Jupyter controller. Spark Master manages the workers in which the executors run. Spark Master's main function is the creation of the RDD (Resilient Distributed Dataset), which is the basic component of Spark. RDD is a distributed and immutable collection of objects that can be rebuilt in case of failure, operations parallelized and distributed through nodes, and data loaded and partitioned through cluster nodes (*executors*).

This component is composed of:

* Legacy code with the binary Spark and its dependencies.
* Integration of the Cassandra driver in the legacy code.
* Spark Master Component Class, following the architectural patterns of the ITI PaaS.
* Master process configuration files.
* The necessary manifests for the ITI PaaS, both service and deployment.
* Communication channels to interact with the ITI PaaS and to link with other components (such as Jupyter, Cassandra).

#### Spark Worker Component

Spark Worker has been integrated into the ITI infrastructure as a software component based on the ITI PaaS. Spark Worker processes the elements assigned by Spark Master and returns the results to the controller. To do this, the workers launch executors to perform the assigned tasks on the data.

This component is composed of:

* Legacy code with binary Spark and its dependencies.
* Integration of the Cassandra driver in the legacy code.
* Spark Worker Component Class, following the architectural patterns of the ITI PaaS.
* Slave process configuration files.
* The manifests required by the ITI PaaS for both service and deployment.
* Communication channels to interact with the ITI PaaS and to link with other components (such as Spark Master, Jupyter, Cassandra).

#### Spark Streaming Component

Spark Streaming has been integrated into the ITI infrastructure as a software component based on ITI's PaaS. Spark Streaming is a component of Spark that enables real-time, high-performance and fault-tolerant processing of streaming data. Spark Streaming enables powerful analytical and interactive applications to be built using both historical and streaming data, while inheriting Spark's ease of use and fault tolerance features. The results of this component (*streaming processing)*, can be transmitted as a data stream through the stream layer or stored directly in the distributed storage layer.

This component is composed of:

* Legacy code with binary Spark and its dependencies.
* Integration of the Cassandra and Kafka controller in the legacy code.
* Spark Streaming Component Class, following the architectural patterns of the ITI PaaS.
* Streaming process configuration files.
* The manifests required by the ITI PaaS, both service and deployment.
* The communication channels to interact with the ITI PaaS and to link with other components (such as Kafka, Cassandra).

### Distributed storage layer

A Big Data storage system groups a large number of servers connected to a high-capacity disk to allow the analysis software to write large amounts of data. The system relies on massive parallel processing databases to analyze ingested data from a variety of sources. In addition, Big Data datasets often lack of structure and come from multiple sources, making them unsuitable for processing with a traditional relational database. The selection of the distributed storage system for the ITI BDAaaS is based on the evaluation of other storage technologies like Hadoop and Cassandra.

Hadoop is open source software written in the Java programming language. It is a framework consisting of a distributed storage system and a set of tools for data analysis. The Apache Hadoop Distributed File System distributes analytical data across hundreds or even thousands of nodes on the server without impacting on performance. Hadoop includes a NoSQL database called HBase using its own HDFS file system. However, Hadoop has a lower performance compared to other NoSQL models. Hadoop consists of two elements, DataNode and NameNode. This last element keeps the directory tree of all files in the file system, and tracks where file data is stored through the cluster. In this scheme, the NameNode is centralized and is a Single Point of Failure for the HDFS Cluster, which could cause availability problems.

On the other hand, Cassandra is a distributed open source NoSQL database based on Java. Cassandra's data model is based on a hybrid model between a value-key management system and a tabular database, the data is organized by table and identified by a primary key, which determines in which node the data is stored. Cassandra was designed to ensure the following characteristics: consistency, availability and partition tolerance. All Cassandra nodes communicate with each other through a peer-to-peer communication protocol called Gossip Protocol that transmits information about the health of the data and nodes. In this way, Cassandra is the best option for a solution that looks for a distributed database that offers high availability and that is also very tolerant to the partitioning of its data when some node of the cluster is offline. For these reasons, Cassandra has been selected as the distributed storage system for the ITI platform.

While Cassandra is a great choice for storing sensor data, there are more efficient ways to work when it comes to storing text strings. The most recommended option used by the community is ElasticSearch. ElasticSearch[[1]](#footnote-1) defines itself as “a distributed RESTful search and analysis engine capable of solving a growing number of usage cases”. ElasticSearch is developed using Java, is open source and distributed under an Apache license. Its main features are:

* Document-oriented (JSON), something common among NoSQL databases.
* It doesn't use schemas, although they can be defined if necessary.
* It is distributed, scaling dynamically and providing high availability.
* All its functionalities are available using the API REST that exposes.
* Clients available for the main languages: Java, JavaScript, Node.js, PHP and Python among others.

Input data, in any format (or no format at all), is transformed during the indexing process into internal documents. These documents have a structure similar to a simple JSON object with keys and values.

#### Cassandra Component

Cassandra has been integrated into the ITI infrastructure as a software component. The main objective of the Cassandra component is to provide a distributed non-relational database with the possibility of configuring the level of information replication between several nodes. This component is interconnected with the other components of the ITI infrastructure, such as Spark, Kafka and Jupyter.

This component is composed of:

* Legacy code with binary Cassandra and its dependencies.
* Cassandra Component Class, following the architectural patterns of the PaaS of ITI.
* Distributed storage configuration files.
* The manifests required by the ITI PaaS, both service and deployment.
* Communication channels to interact with the ITI PaaS and to link with other components.

#### ElasticSearch Component

ElasticSearch has been integrated into the ITI infrastructure as a software component. The ElasticSearch component is primarily intended to provide a distributed search and analysis engine focused on text strings. This component is interconnected with the other components of the ITI infrastructure, such as Spark and Kibana.

This component is composed of:

* Legacy code with the binary ElasticSearch and its dependencies.
* ElasticSearch Component Class, following the architectural patterns of the ITI PaaS.
* Distributed storage configuration files.
* The manifests required by the ITI PaaS, both service and deployment.

### Streaming layer

In recent years, a number of important technologies have emerged that facilitate the connection of data sources to stream processing frameworks in real time. In scenarios where data constantly from the source to the network, and sometimes back to the source, this brings new challenges. For example, if the streaming application that processes the streams were dropped for 2 minutes for some reason, what would happen to the data stream generated during those 2 minutes? Would the data of those 2 minutes be lost? In this sense, Kafka is a good option for dealing with this type of situation and is an appropriate technology for the ITI infrastructure.

Apache Kafka is a distributed publish-subscribe messaging system designed to mainly support the sending of messages with persistence and high performance. One Kafka cluster handles all activity data from different sources. This provides a single data source for consumers both online and offline. This layer acts as a buffer between live activity and asynchronous processing. Kafka can be used to replicate all data in a different data center for offline consumption. Kafka also plays an important role in any streaming application that uses it as an additional storage layer, which stores incoming streaming data and prevents any data loss. If something goes wrong in the Spark Streaming application or in the target database, messages can be retrieved from Kafka. Once the streaming application extracts a message from Kafka, the acknowledgement is sent to Kafka only when the data is replicated in the streaming application.

#### Kafka Component

Kafka has been integrated into the ITI infrastructure as a software component. Kafka is used to build real-time streaming data pipelines that reliably connect data from sources (such as sensors) and the distributed processing layer (stream processing output), to the distributed storage layer and external entities. This component takes into account the main APIs of Kafka:

* Consumer/Producer, which allows an application to subscribe/publish topics, by Kafka;
* Connector, which allows to build and run reusable producers or consumers that connect Kafka topics with existing database systems or applications.

This component is composed of:

* Legacy code with the binary Kafka and its dependencies.
* Kafka Component Class, following the architectural patterns of ITI's PaaS.
* Kafka Consumer/Producer Component Class, following the architectural patterns of ITI's PaaS.
* Kafka Connector Component Class, following the architectural patterns of ITI's PaaS.
* Streaming layer configuration files.
* The necessary manifests for the ITI PaaS, both service and deployment.
* Communication channels to interact with the ITI PaaS and to link with other components such as Cassandra and Spark Streaming.

## ITI BDAaaS Architecture

Figure 15 shows the deployment of the ITI BDA platform based on the functional architecture and technologies selected above. These technologies can be summarized as follows:

* User processing interface: Interactive report or notebook, available to data scientists and analysts. Both Jupyter and Kibana web applications are used.
* Distributed processing: Apache Spark is used to perform batch data processing and streaming (Spark streaming).
* Distributed storage layer: Both Apache Cassandra and ElasticSearch are used to store received data and analysis results.
* Streaming layer: Apache Kafka is used to build real-time streaming data pipelines that receive data from external sensors or stream processing and inject it into the distributed processing layer, into the distributed storage layer or are transmitted as data streams.

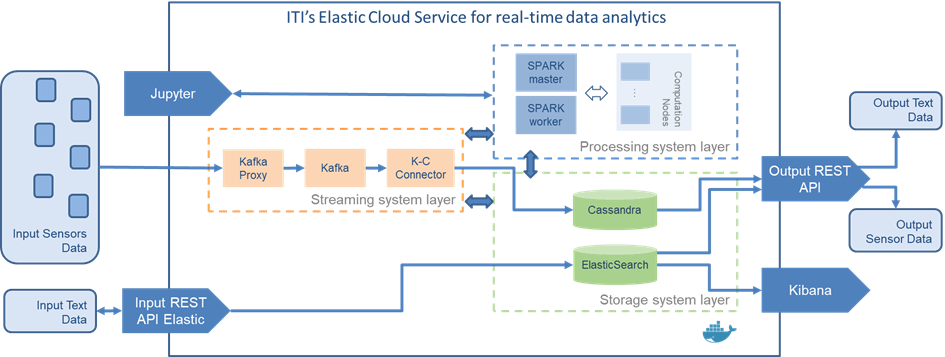


Figure 15 Big Data Analytics Infrastructure of ITI

In order to adapt ITI's Big Data Analytics platform to meet the needs of the Emospaces use cases and interact with other actors, a data exit point has been defined using a specific API. This API could be used by other actors or external entities to access the data collection stored on the ITI platform.

ITI's BDA infrastructure is suitable for addressing data analysis in the various scenarios and environments considered in Emospaces.

# Conclusions

This deliverable contains a summary of the architecture proposed at the beginning of the project, which has been refined during its execution to adapt to the specific needs of the use cases. In addition, each of the different sections of the document describes which components have been chosen to turn the conceptualization represented in the architecture into an executable reality. As a result of this process, the ITI has proposed a platform to give support to the execution of the different use cases. This platform offers a set of tools as a common framework for interaction and functionality. This set of tools can be adapted according to the needs of each of the use cases present in the project.

1. Extracted from its website, https://www.elastic.co/products/elasticsearch [↑](#footnote-ref-1)