Project Acronym:	**SatisFactory**
Project Full Title:	**A collaborative and augmented-enabled ecosystem for increasing satisfaction and working experience in smart factory environments**
Grant Agreement:	**636302**
Project Duration:	**36 months (01/01/2015 - 31/12/2017)**

DELIVERABLE D4.4

Data aggregation toolkit and control middleware that integrates social collaboration and sensing environments

Deliverable Status:	**Final**
File Name:	**SatisFactory-D4.4-v1.0-Data aggregation toolkit and control middleware that integrates social collaboration and sensing environments.pdf**
Due Date:	**August 2016 (M20)**
Submission Date:	
Deliverable Leader:	**FIT**

Dissemination level	
Public	*X*
Confidential, only for members of the Consortium (including the Commission Services)	

This project has received funding from the European Union's Horizon 2020
Research and innovation programme under Grant Agreement n°636302

The SatisFactory project consortium is composed of:		
CERTH[1]	Centre for Research and Technology Hellas	Greece
SIGMA[2]	Sigma Orionis SA	France
FRAUNHOFER	Fraunhofer-Gesellschaft zur Foerderung der Angewandten Forschung E.V	Germany
COMAU	Comau SPA	Italy
EPFL	Ecole Polytechnique Fédérale de Lausanne	Switzerland
ISMB	Istituto Superiore Mario Boella sulle tecnologie dell’informazione e delle telecomunicazioni	Italy
ABE	Atlantis Engineering AE	Greece
REGOLA	Regola srl	Italy
SUNLIGHT	Systems Sunlight Industrial & Commercial Company of Defensive, Energy, Electronic and Telecommunication Systems S.A.	Greece
GlassUP	GlassUp srl	Italy
QPLAN	Q-PLAN International Advisors LTD	Greece

[1] Project Coordinator

[2] Terminated beneficiary since June 2016 and replaced by QPLAN

Authors List

Leading Author (Editor)				
Surname		First Name	Beneficiary	Contact email
Rasheed		Hassan	FRAUNHOFER	hassan.rasheed@fit.fraunhofer.de
Co-authors (in alphabetic order)				
#	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1	Krinidis	Stelios	CERTH	krinidis@iti.gr
2	Lithoxoidou	Eirini	CERTH	elithoxo@iti.gr
3	Sottile	Francesco	ISMB	sottile@ismb.it
4	Tovar	Orlando	ISMB	tovarordonez@ismb.it
5	Tropios	Pantelis	CERTH	ptropios@iti.gr
6	Zikos	Stylianos	CERTH	szikos@iti.gr
7	Ziogou	Chrysovalantou	CERTH	cziogou@cperi.certh.gr

REVISION CONTROL

Version	Author	Date	Status
0.1	FIT	July, 2016	Initial Outline
0.2	FIT	July, 2016	High Level Architecture, middleware services
0.3	ISMB	August, 2016	Device Managers from ISMB
0.4	ISMB	August, 2016	Integration and deployment status of device managers
0.5	CERTH	August, 2016	CERTH contribution for PDEC
0.6	CERTH	August, 2016	Added sections 3.3 and 3.5
0.7	FIT	August, 2016	Section 2.5 is updated
0.8	FIT	August, 2016	Reviewers remarks and comments are incorporated
1.0	FIT	August, 2016	Final version ready for submission

REVIEWERS LIST

List of Reviewers (in alphabetic order)				
#	Surname	First Name	Beneficiary	Contact email
1	Stefanos	Kanidis	SUNLIGHT	s.kanidis@sunlight.gr
2	Vamvalis	Cosmas	ATLANTIS	vamvalis@abe.gr

TABLE OF CONTENTS

List of Figures

List of Tables

List of Definitions & Abbreviations

Abbreviation	Definition
AR	Augmented Reality
CIDEM	Common Information Data Exchange Model
ICT	Information & Communication Technology
IoT	Internet of Things
SOA	Service Oriented Architecture
WP	Work Package
API	Application Programming Interface
DM	Device Manager
GW	Gateway
P2P	Peer to Peer
QoS	Quality of Service
SOAP	Simple Object Access Protocol
SSN	Smart Sensor Network
UWB	Ultra-Wide Band
FP6	Framework Program 6
WSDL	Web Service Description Language
XML	Extensible Markup Language
PDEC	Plant Data Exchange Component
RWTS	Robust Wireless Temperature Sensor
SCADA	Supervisory Control and Data Acquisition Systems

EXECUTIVE SUMMARY

The present document is a deliverable of the SatisFactory project, funded by the European Commission's Directorate-General for Research and Innovation (DG RTD), under its Horizon 2020 Research and innovation programme (H2020), describing the new architecture, improved design, current development status, and prototype implementation of LinkSmart middleware and its core services. It outlines a new architectural approach for different usage and deployment scenarios. It follows a detailed description of the proposed design and core middleware services and their interfaces. An event aggregation component for storing devices and services notifications into CIDEM repository service is also presented. Furthermore, an introduction to available Smart Sensor Networks (sensors, wearable devices, etc.) is given; corresponding Device Managers and their integration with middleware for event notification and collaboration are briefly explained.

1 INTRODUCTION

This chapter outlines the purpose, background, and context of the deliverable as well as the structure of the remaining document.

1.1 *PURPOSE, CONTEXT AND SCOPE OF THIS DELIVERABLE*

In the context of SatisFactory work package structure, Task 4.4 (Development of intelligent notification and control middleware) deals with simplified integration of diverse set of sensors and devices into the SatisFactory framework with of help of LinkSmart middleware. Whereas Task 4.5 (Integration of social collaboration and sensing environments for enhancing collective awareness) is responsible for the development of Device Managers for Smart Sensor Network (SSN) and wearable devices to form a unified service and delivery layer for collective awareness and social collaboration applications. This deliverable is related to these two tasks, and the technical descriptions and specifications provided in this document are subject to change during the course of the project and will be reflected in the final M30 deliverable.

This document is structured as follows. Section 2 introduces the LinkSmart middleware, its high level architecture, and core middleware services with their API descriptions (Section 2.4). An approach for device integration along with design concepts and used terminologies is also presented (Section 2.3). Section 3 presents the Smart Sensor Networks and the Device Managers developed for integrating smart objects, wearable devices, automation systems and IoT (Internet of Things) sensors into LinkSmart middleware. Their deployment status onto the pilot sites is also given.

1.2 *BACKGROUND*

The SatisFactory Platform aims to make traditional factories more attractive, supported by continuous training of their employees, stimulating team interactions and capitalising the created knowledge and experience in every level of their organization. The SatisFactory Platform collects, aggregates and analyses real-time, or near real-time, data from heterogeneous resources, privacy preserving infrared and depth cameras deployed in a factory setting. This gathered knowledge is efficiently utilized to improve the well-being of both the employees and organizations.

Middleware is one of the key elements of SatisFactory Platform as it integrates heterogeneous resources, wearable devices and ICT systems that the platform services build upon. In this integration context, it provides modelling abstractions of real-world ICT data sources, smart devices and sensor systems; and enables search and discovery of these devices and their resources by platform applications and services. To form a service layer for interaction among devices, sensor environments, social collaboration, AR glasses and novel HMIs, middleware is expected to provide unified APIs and protocols for historical and (near) real-time data access.

The SatisFactory Platform uses the LinkSmart open Source middleware[3]. It integrates existing automation systems, sensor networks, and augmented reality devices with their solutions to be developed within the scope of the project. It has been improved and extended by introducing a new and clean architecture, and use of standard and light-weight protocols and technologies. It is complemented with SatisFactory specific components according to the requirements of the project and its applications.

[3] http://linksmart.eu

2 LINKSMART MIDDLEWARE

In this section, an introduction of the LinkSmart middleware is given. A completely new approach to deployment architecture and device integration is elaborated that a follows a brief description of core middleware services.

2.1 *OVERVIEW*

The LinkSmart middleware has its origins in the FP6 IP Hydra[4]. It was developed under the lead of Fraunhofer FIT. After the project ended, the resulting Hydra middleware was renamed to LinkSmart and it was released open source. From that time on, the middleware has been continuously enhanced by different international partners, among them FIT and ISMB. LinkSmart has been evaluated in several applications in ubiquitous computing domains including eHealth and (Energy-aware) Smart Homes [1] [2] [3] [4].

LinkSmart used a service-oriented architecture (SOA) approach in order to offer support across platforms and network boundaries. This higher level generalizes core features, such as security, trust and reflective properties. Developers can easily use web service interfaces for controlling any kind of network technology, such as Bluetooth, RF, ZigBee, RFID, Wi-Fi, etc. As a result, application developers handle the heterogeneity of physical devices with SOA based abstraction layer. Further features of LinkSmart are secure peer-to-peer (P2P) communication, device and service discovery, and respective developer tools.

Main components of the LinkSmart middleware are called managers. A LinkSmart manager encapsulates a set of operations and data that realize a specific functionality. Each connected device runs a Network Manager (NM), which registers all services belonging to the device. All communication between devices happens through the Network Managers. So, network services are accessed through their own Network Manager. The Network Managers are connected to Security Managers (SM) and Crypto Managers (CM), which secure, encrypt and decrypt the communication.
From a deployment prospect, LinkSmart distinguish between devices, those can directly run the middleware, and those which are too resource-constrained to run the middleware. The latter ones are connected via a dedicated gateway device to the LinkSmart network. These gateways have a wired or wireless connection to the resource-constrained device. The gateways run Device Proxies, which handle the communication to the resource-constrained device and translate it to LinkSmart. From the service point of view, the communication with a proxy is identical to a communication with a native LS device. Figure 1 shows a typical LinkSmart deployment including components and interactions. It includes six LinkSmart connected devices: three of them can directly run LinkSmart; other three are connected to a same LinkSmart gateway.

[4] http://hydramiddleware.eu

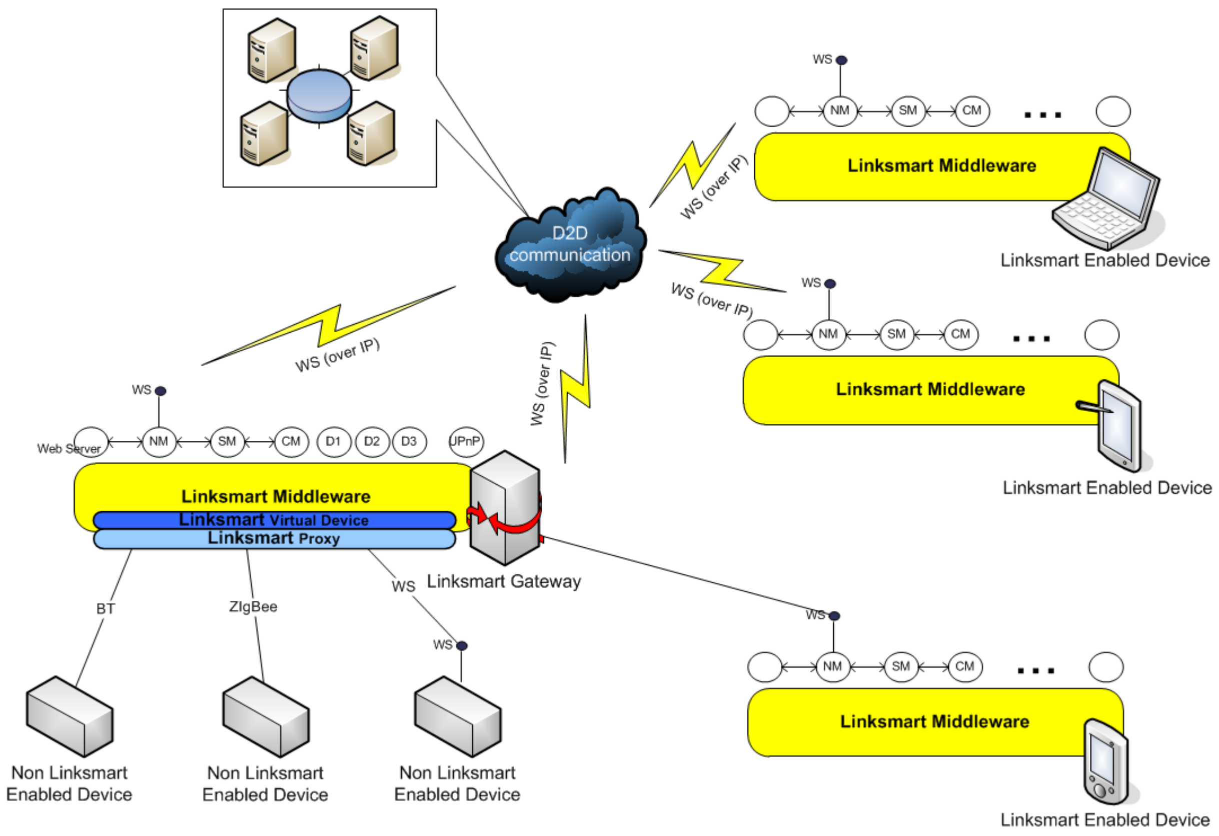


Figure 1 - LinkSmart Deployment Scenario

2.2 *HIGH LEVEL ARCHITECTURE*

LinkSmart has been used and adapted in many EU projects (ebbits[5], Seam4us[6], Impress[7], Adapt4EE[8]) as middleware for connecting heterogeneous devices from different application domains. Based on the gained experience and lessons learned from these projects, LinkSmart is continuously evolving to facilitate and accelerate the development of heterogeneous device environments and service delivery platform. LinkSmart architecture and overall design has been revised based on the following "design objectives":

- **Separation of Concern:** distinguish between deployment and administrative domains based on the usage scenarios
- **Improved Discovery:** simplified device and service discovery in a local and overlay network

[5] http://ebbits-project.eu

[6] http://seam4us.eu

[7] http://impressproject.eu

[8] http://adapt4ee.eu

- **Simplified Device Integration:** an abstraction layer supported by open and standard protocols without relying on some middleware specific Device API abstraction
- **Light-weight Protocols:** enable communication with physical layer by using open and light-weight protocols in a given application domain
- **Support constraint hardware:** can run on a constraint environments like Raspberry or mobile

On an architecture level, LinkSmart has been split into GlobalConnect (GC) and LocalConnect (LC). As name suggests, LC deals with heterogeneity of devices and protocols in a local IP based network. The components constitute the LC environment are shown in Figure 2. Device Managers are software components implementing the device integration for diverse set of devices and ICT systems being integrated as data sources in the middleware. Resource Catalog provides a registry of integrated data sources, their basic meta-information and deployment configuration, including information on how their data can be accessed. Service Catalog provides a registry of middleware and Platform services and can be used for discovery of services by applications using attribute queries. Event Manager provides a message bus for efficient asynchronous communication of Smart Sensor Network (SSN) data streams implementing the Publish/Subscribe communication pattern.

Whereas, GC helps to connect remote LC environments over the internet to form an overlay network into global network as shown in Figure 3. These LC environments can be discovered dynamically. GC provides a tunneling service that enables transparent communication of applications and services beyond the boundaries of a private network.

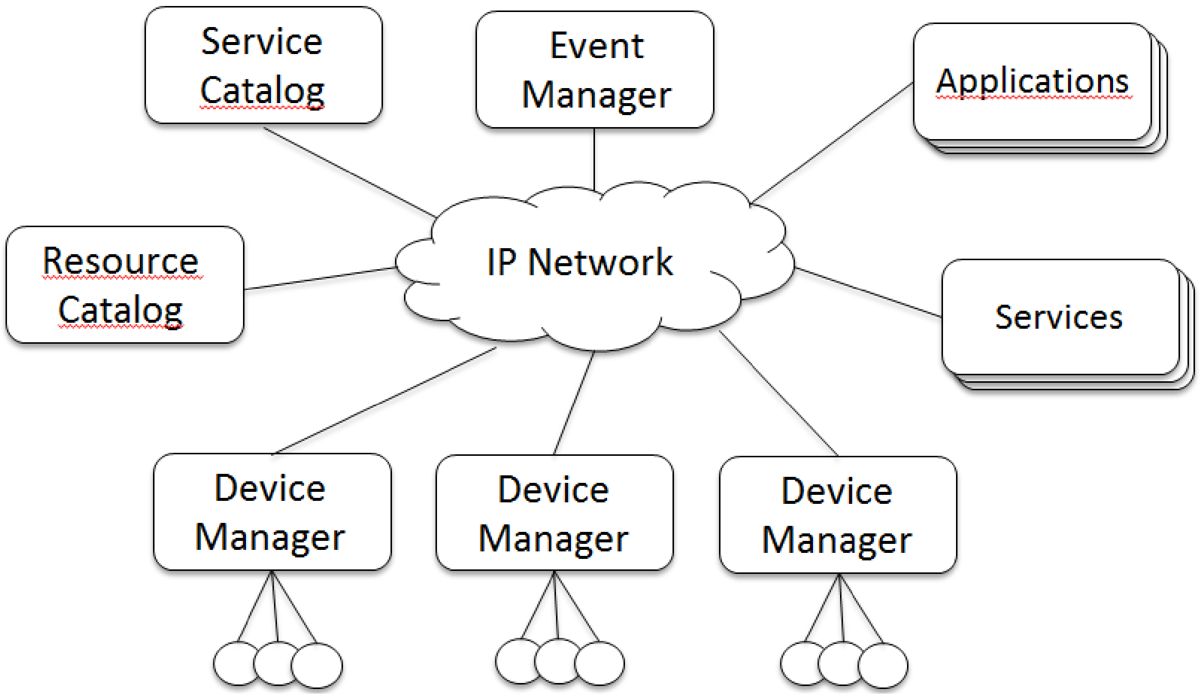


Figure 2 - LocalConnect (LC)

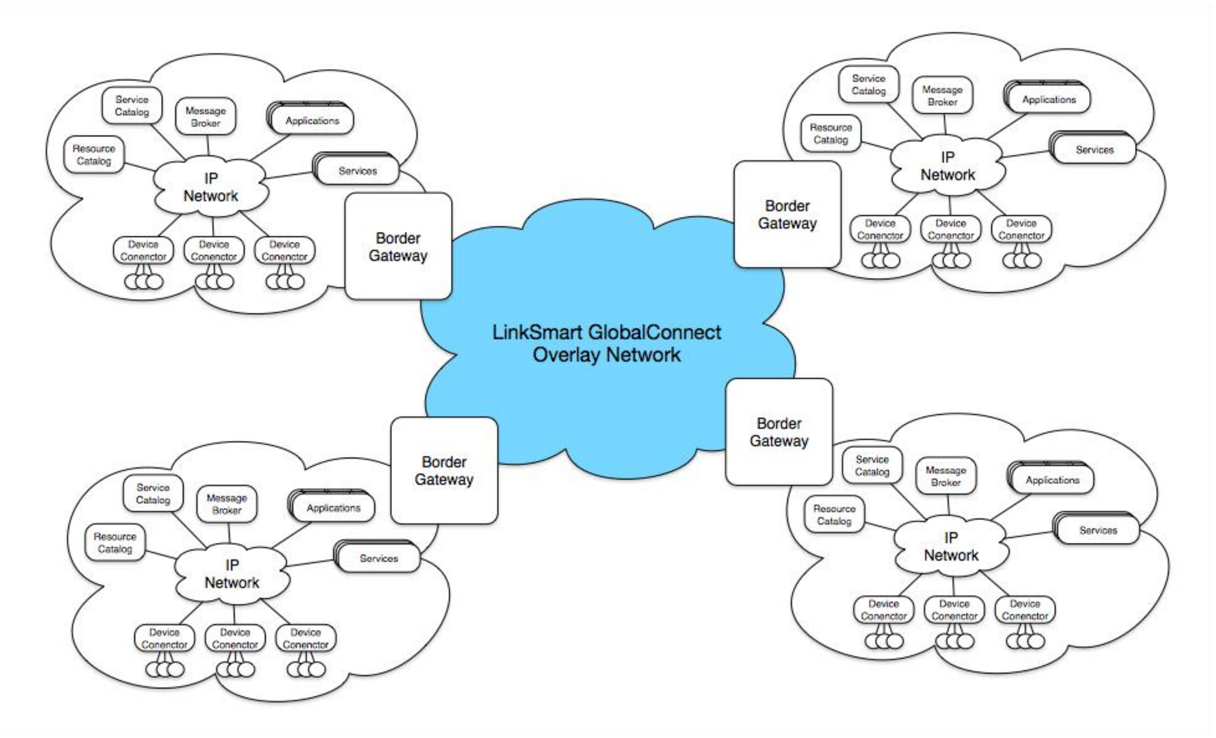


Figure 3 - GlobalConnect (GC)

From an architectural point of view, LinkSmart was originally based on SOA using SOAP web services. However, based on the recommendation by IoT-A[9] and successful experience of adopting a web-based approach in projects like ALMANAC[10] and E3[11], LinkSmart now uses REST as the core architectural principle and RESTful Web Services as the main approach to implement core middleware services. In a similar manner, the XML based WSDL service descriptions are replaced with lightweight JSON data model. Likewise, event notifications from devices are now encapsulated in a lightweight MQTT protocol (detailed description in sub-section 2.4.3) instead of SOAP based envelop. Discovery in local environments is supported by DNS-SD instead of UPnP. As a result, LinkSmart can now be run on constraint hardware like Raspberry or a low resource platform like mobile.

It is important to mention that the SatisFactory Platform uses only LC as determined by project requirements and identified business scenarios. Therefore, in the following sections, we describe the LC concepts and models used by the middleware components, as well as the current status of their design and implementation in more details.

[9] http://www.iot-a.eu

[10] http://almanac-project.eu

[11] http://www.e3-produktion.de/

2.3 DEVICE INTEGRATION

Device Integration addresses the heterogeneity of devices and data sources by providing their unified abstraction to application and services using the SatisFactory Platform. This abstraction is based on the Device and Resource model as explained in next Section. This model is then used for integration of SSN/IoT devices by Device Manager as described in Section 2.3.2.

2.3.1 *Device and Resource Model*

The abstraction for different sensor and ICT technologies integrated via middleware needs to be able to efficiently capture their semantic heterogeneity while preserving a single unified concept and representation. A number of past and ongoing research projects are addressing this problem by classifying diverse IoT data sources using taxonomies and Semantic Web ontologies and standards [5][6]. Other projects and commercial platforms take a more pragmatic approach by using lightweight object models and adopt popular standards on the Web [12]. LinkSmart uses a hybrid approach offering both options to its applications and services by having multiple abstraction layers managed by different middleware services.

In this section, we focus on the bottom abstraction layer that uses an object model to describe the basic meta-data, communication protocols, and deployment configuration of the IoT devices. An object diagram depicting this model is shown in Figure 44.

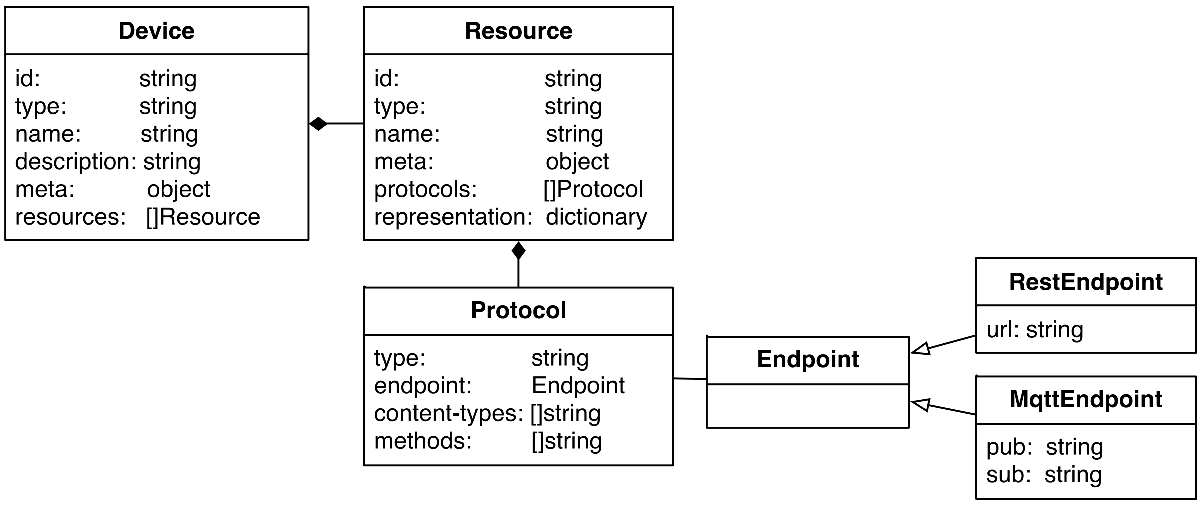


Figure 4 - Device and Resource object model

The model includes the following classes:

[12] http://postscapes.com/internet-of-things-platforms

- **Device** is the top-level class describing an IoT device. In includes basic information about the device such as its human-readable name and description, as well as the **Resources** associated with it.
- **Resource** is an abstraction of a specific resource provided by an IoT device and includes human-readable information about this resource, the supported representation formats, as well as the **Protocols** to access it.
- **Protocol** describes a communication protocol that can be used to obtain and manipulate the state of its **Resource**, including the information about the protocol such as supported methods, content-types of the payload, and **Endpoint**.
- **Endpoint** is the deployment information about the physical endpoint that can be used to access the **Resource** using the communication protocol described by the parent **Protocol** object.

As it follows from the description above, **Devices** and **Resources** are the main concepts that describe heterogeneous IoT devices and sensor systems. The distinction between them is the following:

- **Device** is a physical device with communication capabilities (or a similar virtual abstraction) providing a "native" communication protocol or an API (also proprietary) and exposes various **Resources** (sensors, actuators) over it.
- **Resource** is an abstraction of a resource hosted by a Device, which has a state that can be obtained and/or manipulated (using one of the supported **Protocols**). The access to this state is provided and managed by the hosting **Device**.

In summary, the described object model captures the basic meta-data of the IoT devices; and deployment configuration, including the communication protocols that can be used to access their resources. The device meta-data includes both human-readable information such as *name* and *description*, as well as an optional *meta* object that can be used to define additional human- and machine-readable meta-information. The information defined in the *meta* object that can be used to build higher-level abstraction models, e.g., by using Semantic Web technologies.

2.3.2 *Device Managers*

It is worth mentioning that the Device and Resource model described in the above section is technology and protocol independent, which enables to describe heterogeneous IoT devices and their resources in a unified way. Because the integrated devices and systems often use proprietary and legacy standards, as well as low-level protocols that cannot be directly used by platform services and applications. To be able to use this model by applications and services to discover and communicate with IoT devices, however, each device and resource needs to be modelled using it and this information needs to be made available by the middleware services.

Enabling the communication with IoT devices via open protocols commonly used by applications and services, as well as to actively populate the information about them using the described modelling framework is the task of Device Managers. More specifically, Device Managers are the software components providing the device integration in the LinkSmart and implementing the following functionality:

- Manage devices and their resources in the Resource Catalog middleware service (see Section 2.4.1):
    - **Publish** registrations to Resource Catalog
    - Continuously **update** these registrations
    - Remove the registrations on devices failures and graceful shutdown of the Device Manager
    - Optionally, expose local read-only Resource Catalog API with managed resources

- Provide communication with devices over the network via standardized protocols
    - Expose APIs of devices/resources via open APIs and protocols (HTTP/REST, MQTT, etc.)
    - Implement native APIs/protocols of devices internally (if needed)

The introduced above functional specification of the Device Manager component leaves open its implementation details, as it largely depends on the sensor technology being integrated. As described in Section 3, number of Device Managers are developed using different implementation technologies for integration of the heterogeneous sensor systems in the SatisFactory Platform. Using different implementations, all Device Managers communicate with the middleware services using technology-agnostic protocols and APIs provided by them.

2.4 *CORE SERVICES*

As depicted in Figure 2, core middleware services provide APIs for applications and services working with diverse data sources, such as their discovery and data access. Moreover, they are built on the models and device integration components described in previous Section and provide common functionality required by applications to work with the integrated data sources.

2.4.1 *Resource Catalog*

The Resource Catalog provides a registry of integrated ICT data sources, their basic meta-information and deployment configuration, including information on how their data can be accessed. It also exposes a lightweight JSON-based RESTful API. The Device Managers are supposed to register the available devices and their resources so that applications and services can discover these devices and learn how to communicate with them.

The object model used by the Resource Catalog API is based on the Device and Resource model, extended with additional attributes and classes. The Resource Catalog API provides CRUD (Create, Retrieve, Update, Delete) for device registrations and search/filtering API for querying registered devices and their resources by different attributes. The Resource Catalog API uses JSON-LD as the serialization format for all responses, which makes it possible to use Semantic Web annotations to describe the device meta-data while preserving the lightweight JSON-based API for non-semantic clients. The API also enables Linked Data support: every device and resource can be uniquely addressed through a URL exposed by the Resource Catalog API.

```
PATH: /path
CONTENT-TYPE: application/ld+json
METHODS: GET
RESPONSE:
{
  "id": "<string>",
  "type": "ResourceCatalog",
  "@context": "/static/ctx/catalog.jsonld",
  "devices": {},
  "resources": [],
  "page": "<number>",
  "per_page": "<number>",
  "total": "<number>"
}
```

Figure 5 - Resource Catalog API: ResourceCatalog

The entry point of the Resource Catalog API returns a representation of the **ResourceCatalog** object in the format shown in Figure 55, where:

- **id** is used throughout the API and describes the location (relative URL) of the returned resource, fulfilling the identifiability REST interface constraint [7]. For entry point, it equals to the path in the API endpoint URL, which is configurable and defaults to **/rc**
- **type** and **@context** are special JSON-LD fields that are used to enable Linked Data support and can be ignored by non-semantic clients using the plain JSON API
- **resources** array holds an array of **Resources**
- **devices** object holds a dictionary of **Devices** keyed by **id**, omitting the associated resources
- **page**, **per_page**, and **total** fields are used for pagination of the **resources** array

```
PATH: /static/ctx/catalog.jsonld
CONTENT-TYPE: application/ld+json
METHODS: GET
RESPONSE:
{
  "@context": {
    "id": "@id",
    "type": "@type"
  }
}
```

Figure 6 - Resource Catalog API: JSON-LD context

The basic JSON-LD context used in the current prototype is shown in Figure 66. It resolves the fields **id** and **type** into JSON-LD keywords **@id** (node identifier) and **@type** (node data type) [8].

```
PATH:/rc/<device-id>
CONTENT-TYPE:application/ld+json
METHODS:GET,POST,PUT,DELETE
RESPONSE:
{
"id":"<string>",
"type":"Device",
"name":"<string>",
"description":"<string>",
"meta":{},
"resources":[],
"ttl":"<number>",
"created":"<timestamp>",
"updated":"<timestamp>",
"expires":"<timestamp>",
"page":"<number>",
"per_page":"<number>",
"total":"<number>",
}
```

Figure 7 - Resource Catalog API: Device

Implementing the identifiability REST interface constraint, each device in the Resource Catalog API is uniquely addressable. The representation of the Device resource is shown in Figure 77, where:

- **id** is a relative URL providing a dereferenceable identifier of the device in the catalog. It is constructed as **/path/device-id**, where:
  - **path** is the path in the catalog API endpoint URL
  - **device-id** is a unique device identifier in the network and the convention is to construct it as a **host-id/device-name**, where **host-id** can be, e.g., a FQDN of the host implementing Device Connector
- **name** is a short string describing the device
- **description** is a human-readable description of the device
- **meta** is a dictionary describing additional meta-information of the device
- **resources** is an array of **Resources** associated with the device
- **ttl** is an integer defining the Time-To-Live of the device registration in seconds
- **created**, **updated**, and **expires** fields are generated by the Resource Catalog and describe TTL-related timestamps
- **page**, **per_page**, and **total** are fields used for pagination

Each **Resource** exposed by a **Device** also can be retrieved via a unique URL, which returns its representation as shown in Figure 88, where:

- **id** is a relative URL providing a dereferenceable identifier of the resource in the catalog. It is constructed as **/path/resource-id**, where:
  - **path** is the path in the catalog API endpoint URL
  - **resource-id** is a unique resource identifier in the network and the convention is to construct it as **host-id/device-name/resource-name**, where **host-id** can be, e.g., a FQDN of the host implementing Device Connector.
- **name** is a short string describing the resource
- **meta** is an object describing meta-information of the resource
- **protocols** is an array of protocols which can be used to access the resource
  - **type** is a short string describing the protocol (e.g., "MQTT", "REST")

  - **endpoint** is an object describing the protocol endpoint (e.g., URL for a REST API)
  - **methods** is an array of protocol verbs (e.g., "GET,POST,PUT,DELETE" for REST, "PUB,SUB" for MQTT)
  - **content-types** is an array of strings representing MIME-types defined in representation
- **representation** is a dictionary describing the MIME-types supported by the resource
- **device** specifies the **id** of the device exposing this resource

```
PATH: /rc/<resource-id>
CONTENT-TYPE: application/ld+json
METHODS: GET
RESPONSE:
{
  "id": "<string>",
  "type": "Resource",
  "name": "<string>",
  "meta": {},
  "protocols": [
    {
      "type": "<string>",
      "endpoint": {},
      "methods": [],
      "content-types": []
    }
  ],
  "representation": {},
  "device": "<string>"
}
```

Figure 8 - Resource Catalog API: Resource

As mentioned above, the REST API of the Resource Catalog includes CRUD to create/retrieve/update/delete device registrations, a read-only API for (Linked Data-enabled) **Resources** they expose, and a simple filtering API to search through the catalog for both of them. The full list of URL endpoints and the methods they support is described below.

Device registrations CRUD:

- **/path** returns all registered devices as a **ResourceCatalog**
  - methods: GET
- **/path/** endpoint for creating new entries in the catalog
  - methods: POST
- **/path/<device-id>** returns a specific device registration given its unique id as a **Device**
  - methods: GET (retrieve), PUT (update), DELETE (delete)
- **/path/<device-id>/<resource-id>** returns a specific resource of a specific registration given its unique id as a **Resource**
  - methods: GET

Filtering API:

- **/path/<type>/<path>/<op>/<value>**
- methods: GET

- **<type>** is either **device**/**resource** (returns a random matching entry in the corresponding format), or **devices**/**resources** (returns all matching entries as **ResourceCatalog**)
- **<path>** is a dot-separated path in the Device or Resource similar to json-path
- **<op>** is one of (**equals**, **prefix**, **suffix**, **contains**) string comparison operations
- **<value>** is the intended value/prefix/suffix/substring of the key identified by **<path>**

The **Resources** returned in the **resources** array of the **ResourceCatalog** and **Device** are paged using the **page** and **per_page** parameters. The results are then including the following additional entries:

- **page** is the current page (if not specified - the first page is returned)
- **per_page** is the number of entries per page (if not specified - the maximum allowed is returned)
- **total** is the total number of Resources in the catalog

Device Managers can use Resource Catalogs available on the network and additionally implement Resource Catalog API locally for the devices and resources they manage. Registrations in different Catalogs must have same ids (prefixed with the path of the corresponding Resource Catalog API entry point). For applications working with a Resource Catalog it must be transparent whether they interact with a local API of a Device Connector or a global Resource Catalog: all of them must implement the same Resource Catalog API described here. Registrations in the remote catalog have a TTL and need to be updated before they are expired and removed. For devices in the local catalog, TTL = -1 and these registrations never expire.

During the remaining period of efforts (till M30), it is planned to introduce authentication and authorization layer for user identification and access rights. A persistence layer for configurable storage backend will be added as well. Furthermore, current API will go under minor revisions as per the experience gained during the development period, and feedback of the partners working on device integration.

2.4.2 *Service Catalog*

The Service Catalog service has similar functionality as the described above Resource Catalog with a difference that it provides a registry for middleware and platform services. Service Catalog enables search and discovery of available platform services by attributes, such that applications and services can dynamically discover required services without prior configuration of the endpoints. In addition to that, the endpoint of the Service Catalog API can be advertised on the network using DNS-SD, which enables automatic system discovery in the SatisFactory platform.

Similarly to the Device and Resource model used by the Resource Catalog to describe heterogeneous IoT devices and resources (and their APIs), Service Catalog uses an object model to describe different services and their capabilities. Just like IoT devices, the services are expected to be different: talk different protocols, use different data formats, etc. The model used by the Service Catalog and the representation formats returned by its API are greatly inspired by those used by the Resource Catalog as described below. The Service Catalog API is also REST/JSON-LD based with CRUD for service registrations and a query by attributes API for clients. Unlike the Resource Catalog, it is expected to have only a “global" Service Catalog in the whole platform deployment.

```
PATH: /path
CONTENT-TYPE: application/ld+json
METHODS: GET
RESPONSE:
{
  "id": "<string>",
  "type": "ServiceCatalog",
  "@context": "/static/ctx/catalog.jsonld",
  "services": [],
  "page": "<number>",
  "per_page": "<number>",
  "total": "<number>"
}
```

Figure 9 - Service Catalog API: ServiceCatalog

The entry point of the Service Catalog API returns a collection of Services as a **ServiceCatalog** object in the format shown in Figure 9, where:

- **id** is used throughout the API and describes the location (relative URL) of the returned resource, fulfilling the indentifiability REST interface constraint [5]. For entry point, it equals to the path in the API endpoint URL, which is configurable and defaults to **/sc**
- **type** and **@context** are special JSON-LD fields that are used to enable Linked Data support and can be ignored by non-semantic clients using the plain JSON API. In the current implementation, the same basic @context as in Resource Catalog (Figure 66) is used
- **services** array holds an array of **Services**
- **page**, **per_page**, and **total** fields are used for pagination of the **services** array

Implementing the identifiability REST interface constraint, each service in the Service Catalog API is uniquely addressable. The representation of the Service resource is shown in Figure 10 where:

- **id** field is a relative URL providing a dereferenceable identifier of the **Service** in the catalog. It is constructed as **/path/ + service-id**, where:
    - **path** is the path in the catalog API endpoint URL
    - **service-id** is a unique service identifier in the network and the convention is to construct it as **hostname/servicename**
- **name** is a short string describing the **Service**
- **description** is a human-readable description of the **Service**
- **meta** is any hashmap describing meta-information of the **Service**
- **protocols** is an array of protocols supported by the **Service**
- **type** is a short string describing the protocol (e.g., "MQTT", "REST")
- **endpoint** is an object describing the protocol endpoint (e.g., URL for a Web API)

```
PATH: /rc/<resource-id>
CONTENT-TYPE: application/ld+json
METHODS: GET
RESPONSE:
{
  "id": "<string>",
  "type": "Service",
  "name": "<string>",
  "description": "<string>",
  "meta": {},
  "protocols": [
    {
      "type": "<string>",
      "endpoint": {},
      "methods": [],
      "content-types": []
    }
  ],
  "representation": {},
  "ttl": "<int>",
  "created": "<timestamp>",
  "updated": "<timestamp>",
  "expires": "<timestamp>"
}
```

Figure 10 - Service Catalog API: Service

- **methods** is an array of protocol verbs (e.g., "GET,POST,PUT,DELETE" for REST, "PUB,SUB" for MQTT)
- **content-types** is an array of strings representing MIME-types defined in representation
- **representation** is a dictionary describing the MIME-types supported by the **Service**
- **ttl** is an integer defining the Time-To-Live of the **Service** registration
- **created**, **updated**, and **expires** are generated by the server and describe TTL-related timestamps
- **page, per_page** and **total** are used for pagination

Services can be exposed through different protocols, and below are conventions for describing some of them (format of entries in the **protocols** array).

MQTT

- **type**: MQTT
- **endpoint**: {"url": "scheme://address:port", "topic": "/topic"}, where:
    - **url** describes the broker connection information as a URL (RFC 3986) using the following URI scheme
    - **topic** is an optional field describing the topic used by the service. If the described service itself is an MQTT broker, can be omitted
    - additional fields can be defined
- **methods**: ["PUB", "SUB"] - array of supported MQTT messaging directions. If the described service itself is an MQTT broker, this indicates whether it supports subscription, publishing, or both.
    - **PUB** - indicates that the service publishes data via MQTT
    - **SUB** - indicates that the service subscribes to data via MQTT

- **content-types**: ["application/json", ...] - array of supported MIME-types. Empty means payload agnostic

REST

- **type**: REST
- **endpoint**: { "url": "scheme://address:port"}, where:
    - **url** describes the endpoint URL
    - additional fields can be defined
- **methods**: ["GET", "PUT", "POST", ...] - array of supported HTTP methods
- **content-types**: ["application/json", ...] - array of supported MIME-types

The Service Catalog REST API provides CRUD to create/retrieve/update/delete **Service** registrations and a simple filtering API to query the catalog for services by attributes. The full list of URL endpoints and the methods they support is described below.

Service Registration CRUD:

- **/path** returns all registered services as a **ServiceCatalog**
    - methods: GET
- **/path/** endpoint for creating new entries in the catalog
    - methods: POST
- **/path/<service-id>** returns a specific service registration given its unique id as a Service
    - methods: POST (create), GET (retrieve), PUT (update), DELETE (delete)

Filtering API:

- **/path/<type>/<path>/<op>/<value>**
- methods: GET
- **<type>** is either **service** (returns a random matching entry) or **services** (returns all matching entries)
- **<path>** is a dot-separated path in the **Service** similar to json-path
- **<op>** is one of (**equals, prefix, suffix, contains**) string comparison operations
- **<value>** is the intended value/prefix/suffix/substring of the key identified by **<path>**

Pagination, versioning and expiration of registrations in the Service Catalog API are identical to those used in the Resource Catalog and described in section 2.4.1.

The Service Catalog API is intended to be used directly by the platform services (also using client libraries) for registration and continuous update of their information. The services should be registered upon initialization and de-registered on shutdown or failure. If not updated within the specified TTL value, the information about service automatically removed from the Catalog. Third-party services can be registered in the Service Catalog using additional components taking care of their registration and continuous update of the registration information.

For future steps and actions, same as Resource Catalog future development actions, an authentication and authorization layer for user identity and access rights will be developed. A configurable persistence backend will also be added. It is also foreseen that API go under some minor changes based on the feedback from project partners.

2.4.3 *Event Manager*

It provides a message bus for efficient asynchronous communication of sensor data streams, where the receiving party registers (*subscribes*) to a pre-defined *topic* and receives new messages as the sending party *publishes* them on that *topic.* It mediates the forwarding of messages from the sender (*Publisher)* to the receiver (*Subscriber).* More specifically, it implements publish/subscribe [9] communication pattern that is used for communication between services and transmission of sensor data in the SatisFactory Platform.

In recent times, the Message Queue Telemetry Transport (MQTT)[13] is recognized as the de-facto standard for Publish/Subscribe communication in the IoT domain. MQTT has been designed at IBM as a messaging protocol for reliable delivery of messages over unreliable networks by constrained devices under high requirements to low latency. MQTT is a simple messaging protocol working on top of TCP. It is a binary, payload-agnostic protocol with minimal overhead designed for unreliable networks with limited bandwidth. It has been then made publicly available and recently standardized by OASIS [10]. As a Publish/Subscribe protocol, MQTT provides a number of features like topic wildcards, different level of quality of service, retained messages, last will and testament, and persistence sessions. A number of Message Broker implementations, as well as client libraries are available under Free and OpenSource licenses and supported by the IoT working group of Eclipse Foundation[14].

Event Manager supersedes the old proprietary event format based on SOAP protocol with MQTT for event notifications from sensor data streams. In addition to forwarding sensor data streams, it is also used by the SatisFactory Platform services for asynchronous communication. Similarly to other services, the Event Manager is registered in the Service Catalog, where it can be discovered by platform applications and services. The high level interaction of Event Manager with various other components of the SatisFactory Platform is depicted in Figure 11; where Device Managers are publishing events from SSN to the Event Manager, and the platform services subscribed for designated topics, are get notified.

[13] http://mqtt.org

[14] http://iot.eclipse.org/

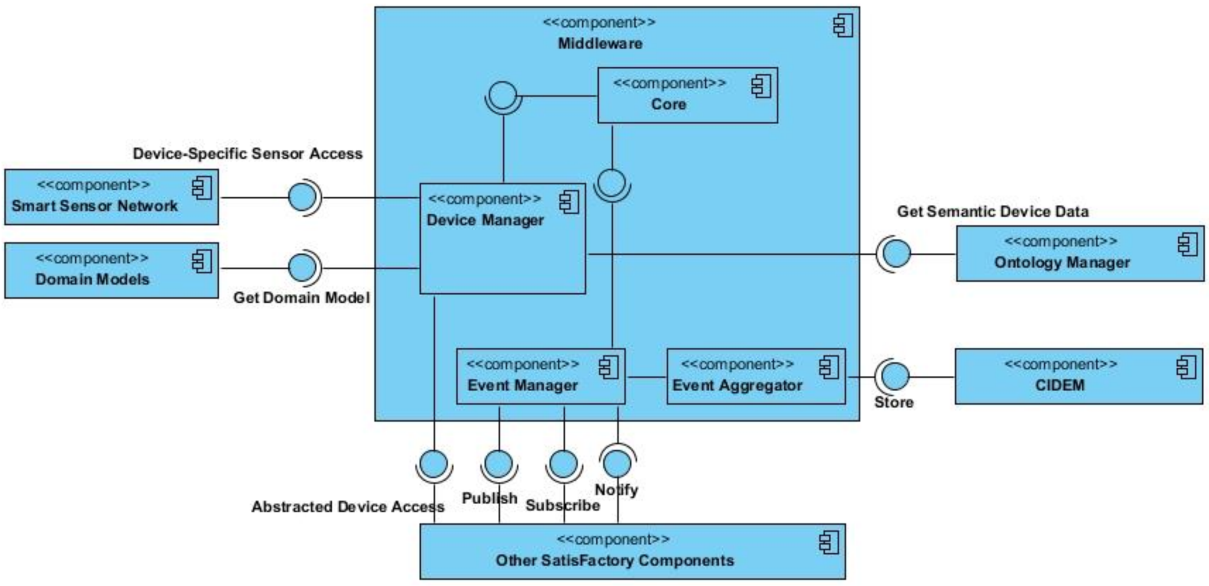


Figure 11 - Event Manager Interaction

The magnitude and velocity of event propagation in a CPERI pilot site is currently being examined. The deployment of Event Manager will support "bridging" to have a high level of availability and scalability in case it is required.

2.5 *Event Aggregator*

The SatisFactory platform services and applications provide actionable knowledge based on the data streams (events and measurements) acquired from Smart Sensor Network and wearable devices. This real-time dynamic information from Smart Sensor Networks is gathered by Device Managers. They publish this gathered data to the Event Manager using MQTT protocol. The CIDEM repository service exposes a REST API for storing this data in a CIDEM compliant form. The Device Managers doesn't need to rely on the REST interface of the CIDEM repository for data storage. Therefore, it is feasible by LinkSmart to collect event notifications from the Event Manager and store them in a CIDEM repository.

Event Aggregator is such a component archiving the events and measurements received from Event Manager to CIDEM repository as shown in Figure 12. As Device Managers are responsible for collecting the data from physical layer components, they register the device object model (see section 2.3.1) in a Resource Catalog. The Device Manager can set a Boolean flag in Meta section of this object model if collected data should be stored in a CIDEM repository service. High-level SatisFactory services (such as DSS, HR re-adaptation toolkit) can also be registered into Service Catalog if some of the processed data should be stored as well in a CIDEM repository service. After registration, Device Managers and services publish data to the Event Manager, from where it gets delivered to designated receivers.

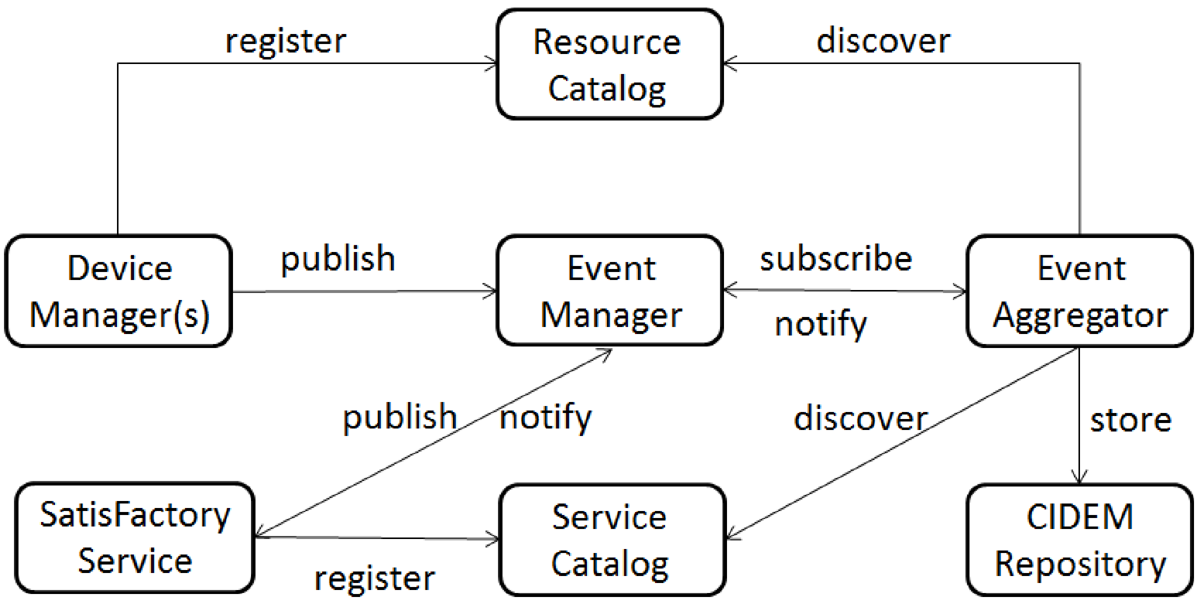


Figure 12 - Event Aggregator Interaction

Event Aggregator queries Resource and Service Catalogs periodically to discover new (and remove old) sensor devices and services. It parses and filters out the device and service registrations to determine the provision for CIDEM storage. With the information about Event Manager and MQTT topic, it subscribes to events and measurements from those devices data and services sources. Upon receiving notification events from Event Manager, it publishes them in the CIDEM repository service.

An initial version of the Event Aggregator has been deployed onto the CPERI pilot site. However, it is foreseen that further improvements are required for handling large data stream and scalability.

3 INTEGRATION OF SMART SENSORS NETWORK INTO LINKSMART

This section presents the integration of the Smart Sensor Network (SSN) into LinkSmart. Furthermore, it presents the integration of the AR In-Factory Platform component into the middleware.

As presented in the deliverable D2.1 [11], the SSN consists of heterogeneous devices that allow the interaction between the physical world (i.e., the shop floor) and the SatisFactory platform. As depicted in the red part of Figure 13, the SSN includes: UWB devices (fixed and wearable), depth and thermal cameras, new IoT sensors equipped with a robust communication module, legacy sensors (made available by the Plant Data Exchange Component (PDEC)) and AR glasses.

According to the LinkSmart definitions, the devices belonging to the SSN group can all be considered as resources.

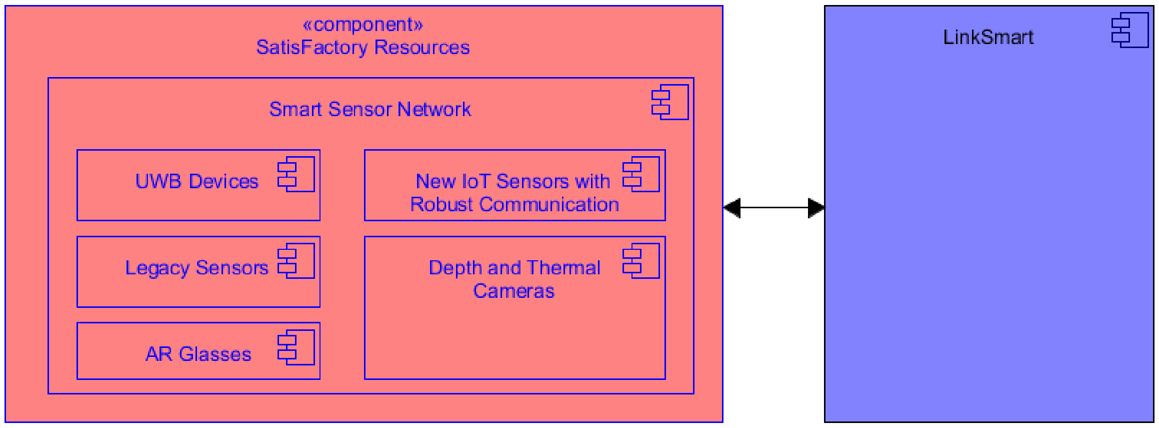


Figure 13 - Smart Sensor Network Integration

It is important to mention that the integration of SatisFactory social collaboration components is planned to start in M21 and final M30 deliverable will document the integration efforts. These components are gamification framework, social collaboration platform and digital andon system. The latter two are used as visual output components for data from the gamification framework. The digital andon system displays the team points of the gamification framework. For this, the gamification framework calls the REST API of the digital andon. The social collaboration platform displays the personal points of the gamification framework. For this, the social collaboration platform calls the REST API of the gamification framework. In both cases digital andon and gamification framework will be registered as LinkSmart Services so that they are discoverable by SatisFactory Platform applications.

3.1 *UWB-BASED WEARABLE DEVICES*

UWB-based-Wearable Devices (UWB-WDs) are hardware platforms developed by ISMB capable of providing localization and ergonomics functionalities to the workers by means of a UWB localization system. In particular, the UWB localization system is composed of three types of devices: tags, anchors and a gateway (GW). Typically, anchors (UWB anchors) are placed in fixed and well know positions at the shop floor. Tags are the UWB-WD (i.e., they are worn by workers whose positions need to be estimated). Each UWB-WD runs a Bayesian algorithm that estimates its own position on the basis of the collected data (i.e., ranging measurements w.r.t. anchors and anchors positions). In addition, the inertial sensors are used to provide more robust location estimation as well as to support ergonomics features. Finally, the UWB gateway (UWB-GW) is the end-point of the localization devices. In particular, it receives localization and ergonomics data from the UWB-WDs and forwards them to the Localization Manager (LM) through the LinkSmart middleware. Further details about hardware design and implementation of the UWB-WDs are presented in the deliverable D4.1 [12] while the design and implementation of the localization algorithm is presented in the deliverable D3.3 [13].

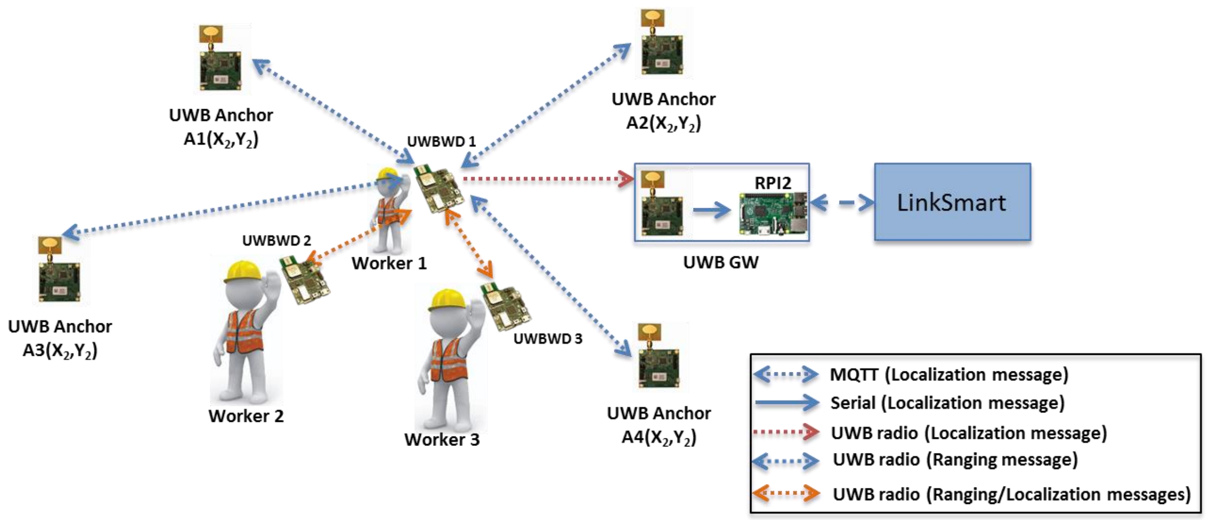

UWB Anchor A1(X2,Y2)
UWB Anchor A2(X2,Y2)
UWB Anchor A3(X2,Y2)
UWB Anchor A4(X2,Y2)
UWBWD 1
UWBWD 2
UWBWD 3
Worker 1
Worker 2
Worker 3
RPI2
UWB GW
LinkSmart
MQTT (Localization message)
Serial (Localization message)
UWB radio (Localization message)
UWB radio (Ranging message)
UWB radio (Ranging/Localization messages)

Figure 14 - Overall UWB Communication Architecture

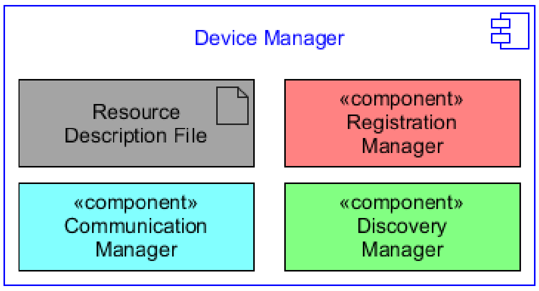
Figure 1414 shows the overall UWB communication architecture. As it can be observed, all UWB devices use a UWB transceiver compliant with the IEEE 802.15.4a standard to communicate and perform ranging measurements. The ranging messages are based on a proprietary format defined by the manufacture of the UWB technology[15] while localization messages are defined ad-hoc by ISMB. As mentioned above, the UWB-GW is the end-point of the localization system; it receives the localization messages from the UWB-WDs by using a UWB transceiver, which is connected to a Raspberry PI2 (RPI2) via a serial COM. The RPI2 runs a component named Device Manager (DM), which allows the integration of the UWB localization system with LinkSmart.

In order to guarantee a correct integration with LinkSmart, the following functionalities have been implemented in the Device Manager:

- publish registration to the Resource Catalog
- continuously update the registration in the Resource Catalog
- remove the registration of the RC when the component is not available
- expose APIs of the resources via open APIs and protocols (e.g., via HTTP/REST, MQTT, etc.)

The architecture of the UWB Device Manager, which is depicted in Figure 1515, has been designed considering the above reported functionalities. As it can be observed from the figure, the UWB Device Manager has a local “Resource Description File” and three software modules, namely “Registration Manager”, “Communication Manager” and “Discovery Manager”, which are presented as follows.

[15] http://www.decawave.com/


Device Manager
Resource Description File
«component» Registration Manager
«component» Communication Manager
«component» Discovery Manager
Figure 15 - Device Manager Architecture

- **Resource Description File (RDF)**: This file contains a description of the UWBGW device and the corresponding resources (i.e. UWBWDs) that it exposes along with their APIs. The contents of the RDF will be registered to the Resource Catalog, thus allowing other components to discover the UWBWDs and learn how to receive localization data from them. In addition, the RDF is registered to the RC with a corresponding Time-To-Live (TTL) parameter. LinkSmart uses this parameter to automatically remove the registered resource from the RC when TTL is expired.
- **Registration Manager (RM)**: This component periodically registers the RDF into the RC every TTL/2 in order to keep alive the registration of the UWBGW into LinkSmart. To do so, the DM uses the RESTful API of LinkSmart. In case the device is not available anymore, the RM is stopped and as consequence its resources automatically will be removed from the RC.
- **Discovery Manager (DM)**: It periodically makes a local copy of the RC and looks for the available resources on the basis of keywords that allow it to identify the resources/services of interest. If a resource is identified, the DM retrieves its data and saves them locally, especially the ones related to the protocols that are used by the component. Some actions will be taken according to the available protocols; for example if a resource/service uses MQTT, the DM will subscribe to its topics. Alternatively, if RESTful APIs are used, the DM will read the methods and use them for receiving or sending data. When it is recognized that the component is not available, the local register for that component is deleted.
- **Communication Manager (CM)**: It forwards the data provided/generated by the UWB-WD into LinkSmart by using MQTT protocol. In addition, the CM uses the device drivers in order to receive the localization messages from the serial port given by the UWB device as depicted in Figure 1616. Further details about MQTT Localization messages are presented in [13].

The UWB-based localization system was deployed and tested at the CERTH/CPERI pilot site at M15. More details about the deployment are reported in the deliverable D5.3 [14] whiles the performance of the system in the deliverable D3.3 [13].

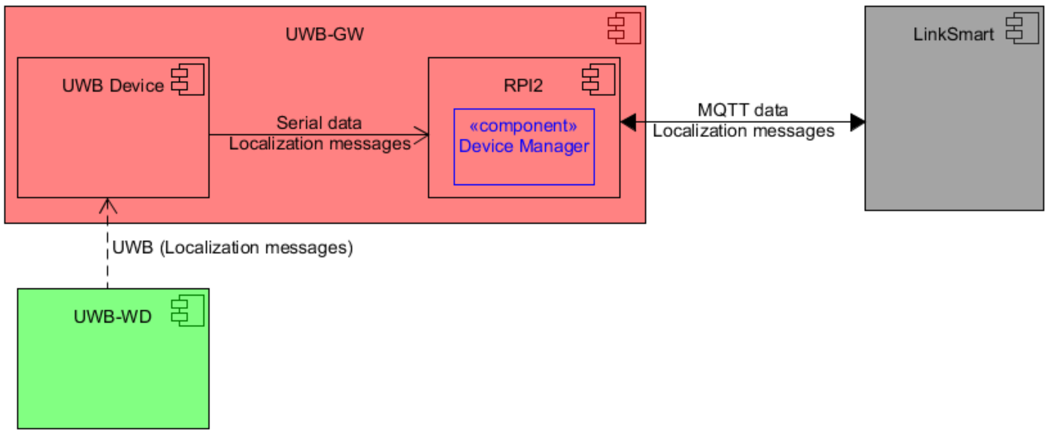


Figure 16 - UWB-GW Integration with LinkSmart

Overall, the first integration with LinkSmart has been completed and tested. During the second half of the project, the effort will be devoted to improve few aspects of the integration, for instance, the update of the Resource Catalog when a UWB-WD is not available. Related results will be presented in the second iteration of this deliverable.

3.2 *Robust Wireless Temperature Sensor*

The Robust Wireless Temperature Sensor (RWTS) is based on the 6LoWPAN communication protocol. As depicted in Figure 17, the RWTS periodically senses the temperature and sends the data to its GW (i.e. RWTS-GW). In turns, the RWTS-GW sends the temperature messages to LinkSmart by means of the MQTT protocol.

Since the wireless communication channel in industrial environments is heavily affected by metallic objects and interference, the RWTS adopts a robust communication scheme based on a frequency agility module described in deliverable D4.1 [12]. This module enables the temperature sensor node to select the best channel for data transfer in case high interference or data loss is detected on the current operating channel. The integration of the RWTS-GW with LinkSmart has been done following the same approach of the UWB-GW's one (reported in section 3.1).

The RWTS was deployed and tested at the CERTH/CPERI pilot site at M15. More details about the deployment are reported in the deliverable D5.3 [14], while the performance of the system is reported in the deliverable D4.1 [12].

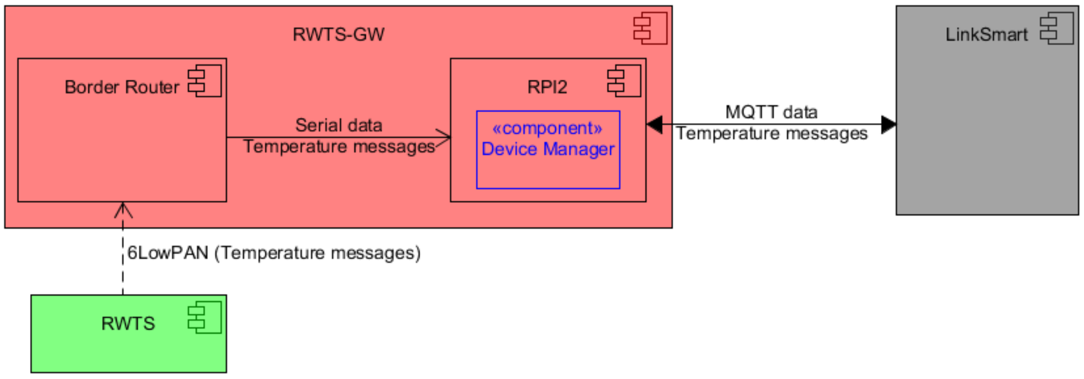


Figure 17 - RWTS-GW Integration with LinkSmart

Overall, the integration with LinkSmart has been completed and tested for RWTS. During the second half of the project, the effort will be devoted to the integration of the RWTS by using a multi-radio technology [12]. Related results will be presented in the second iteration of this deliverable.

3.3 *Comfort Level Measurement Sensors*

The Comfort Level Measurement Sensors are utilized across the CERTH/CPERI shop floor at places where usual human occupancy is to be expected. The sensors are periodically measuring ambient temperature, humidity and lightning conditions per deployment area and are publishing the measurement data via MQTT messages to LinkSmart.

There are two types of comfort level measurement sensors. The main system is based on a ZigBee robust sensor network. ZigBee sensors are battery based and can achieve months or even years of battery life, mainly based on the chosen reporting period. ZigBee sensors gather their data on their ZigBee coordinator who also acts as a global ZigBee data sink. The ZigBee coordinator is combined with an ESP8266 Wi-Fi module to form the ZigBee gateway which interconnects the ZigBee network with the IP based network in order to post measurements received from ZigBee sensors to LinkSmart as shown in Figure 18. For remote locations where ZigBee signal is out of reach while there is Wi-Fi coverage and mains power availability, standalone Wi-Fi sensors also based on the ESP8266 SoC can be deployed. The later ones can directly post measurements to LinkSmart via MQTT.

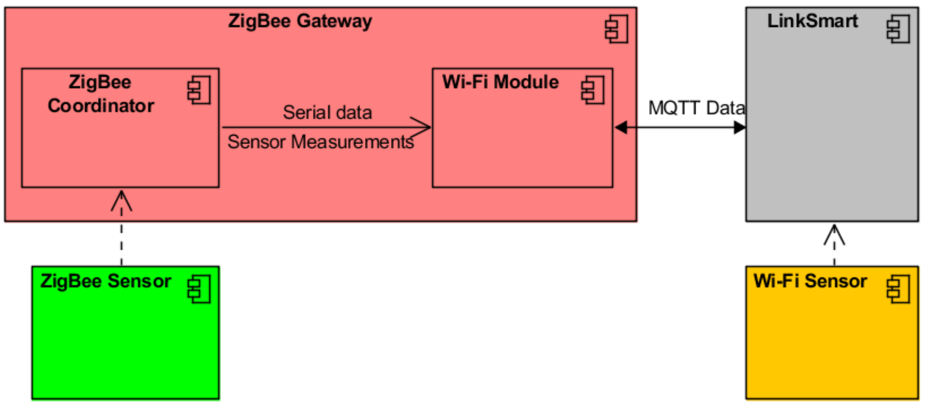


Figure 18 - Comfort Sensor Network Integration with LinkSmart

Firmware for the ESP8266 SoC is developed by using the Arduino IDE. In order to implement the integration with LinkSmart, the open source PubSubClient library which offers MQTT publish and subscription support is going to be deployed on both the ZigBee gateway and the Wi-Fi based sensors.

Currently a prototyping Wi-Fi based sensor is deployed and tested at the CERTH/CPERI pilot site. LinkSmart integration is not yet implemented. Deployed sensor is serving data via a web interface. Firmware is currently being developed to implement integration with LinkSmart.

3.4 *AUTOMATION SYSTEM - LEGACY SENSORS*

The Automation Systems that are present at CERTH/CPERI shop floor provide the data for the communication with the pilot plants and continuous processes where the Legacy Sensors are present. The Automation Systems are used for the process control and remote monitoring that provide their input to Supervisory Control and Data Acquisition Systems (SCADA) for chemical and energy production processes that are in operation with respective middleware software, with OPC connectivity. A number of multi-sensorial network of heterogeneous distributed systems (indicative industrial protocols: CANBus, Profibus, EtherCat, RS485) are involved at Satisfactory project.

CERTH/CPERI's shop floor consists of numerous industrial process systems where information is propagated through a network backbone that facilitates data exchange from and to the input/output field of respective units. Each unit has sensors that acquire various signals which are distributed in a wide area and are connected using industrial networks. In order to provide a uniform way of communication at the application layer between the various sources and sinks of data, OPC-based interfaces are developed and deployed at each individual process unit.

A Plant Data Exchange component (PDEC) is developed in order to concentrate and make available to SatisFactory platform the data for the I/O field of the process systems. This software component supports the transmission and transformation of the shop-floor data to ISA-95 complaint format. It combines diverse data from shop-floor automation systems and the IoT sensors to support decision making, Also PDEC is responsible for providing feedback to the existing HMIs that the workers currently use to interact with the processes.

PDEC communicates with the automation systems (SCADA), filters the data and categorizes them before sending the necessary information, notifications and alert to the other components of SatisFactory platform.

PDEC provides data to three components of the SatisFactory platform using respective number of APIs according to the requirements of the corresponding activities performed by the components (Middleware, CIDEM, iDSS). A summary of the communication capabilities is presented in Table 1.

Table 1 - PDEC and communication with SatisFactory components

Component	***API***	***Comments***
Middleware (LinkSmart)	MQTT Protocol	Receives alarm information from the shop floor in real time
Common Information Data Exchange Module (CIDEM)	RESTful Services	Receives all type of data (nominal values, alarms) from the Pilot Plants in real time
Integrated DSS	RESTful Services	Receives alarm information from the shop floor in real time

The information that is available to components through the Linksmart can access data of the PDEC components at the topic "/device/automationsystem".

The communication and testing data transmission has been deployed and validated by CERTH with the three aforementioned components (CIDEM, iDSS, Middleware).

- The Site Acceptance Tests (SAT) for the CIDEM have been completed at M15 (March 2016) and the PDEC has been commissioned to be used at M15
- The SAT for the iDSS have been completed at M16 (April 2016) and the PDEC has been commissioned to be used at M16 with the iDSS
- Initial testing and sample data transmission to Linksmart has been performed at M18 (June 2016) and by M20 (August 2016) data related to the alarms and nominal values are sent at the predefined sampling intervals

After the final commissioning and steady deployment of the middleware-related components the direct communication of the PDEC with the iDSS and CIDEM will be stopped and all communication will take place through the Linksmart. The related activities are expected to take place during M21-M22, in order to have a fully functional system by M23 (November 2016). These activities will be performed by the collaboration of FIT with CERTH and ABE.

3.5 *Depth and Thermal Cameras*

Depth and Thermal cameras are utilized for the detection of incidents of different nature such as falls, falling items, collisions and intrusions to restricted areas of the shop-floor and overheated electric/chemical equipment respectively. Once an incident is detected an alarm is set so that the appropriate coping mechanism is triggered. The Depth and Thermal Sensor Network's deployment follows client- server architecture. Each depth/thermal sensor is connected with a separate Client-PC.

The depth cameras utilize a USB connection. The clients forward the information collected from the cameras to a server via TCP/IP connection where the actual correlation of received data takes place providing the necessary information for the incident detection manager that operates at the same server. On the contrary, the thermal cameras use a TCP/IP connection to the Client-PC. The client-PCs are responsible for data processing and to provide all relevant incident-related information to the incident detection manager. Other components of the SatisFactory platform are notified of the occurring incidents through middleware, i.e. Linksmart, using MQTT protocol. The transferred alarm message includes information referring to the type and location of the occurring incident.

The communication and testing data transmission was deployed and validated by CERTH using Middleware component at M19 (July).

3.6 *AR Glasses*

AR Glasses are visual devices worn by operators to help them on their daily job by overlapping to reality useful information coming from the main information system of the company. Their main functionalities are to present information or alerts to the operator in his field of view, to enable audio communications with remote senior workers, to raise an alarm in case of high level of noise within the working environment, to stream video from an on board camera to remote workers and to show objects' temperature in the field of view by means of an on board thermal camera.

In general, the AR Glasses are controlled by the AR In-Factory Platform component as depicts Figure 19. In particular, the AR Platform communicates over WIFI with AR Glasses by making use of RESTful services with CIDEM compatible XML format messages. More specifically, the RESTful services are used to show the operator data and alerts coming from the SatisFactory environment as well as to manage camera, thermal camera and the other on board sensors. The integration of the AR Glasses with the rest of the SatisFactory components is achieved through LinkSmart middleware and AR In-Factory platform. Alerts and other important messages are send to the AR Glasses through middleware (utilizing the specific MQTT/RESTful services), while AR information for Standard Operative Procedures (SOPs) and scenarios are sent through AR In-Factory platform.

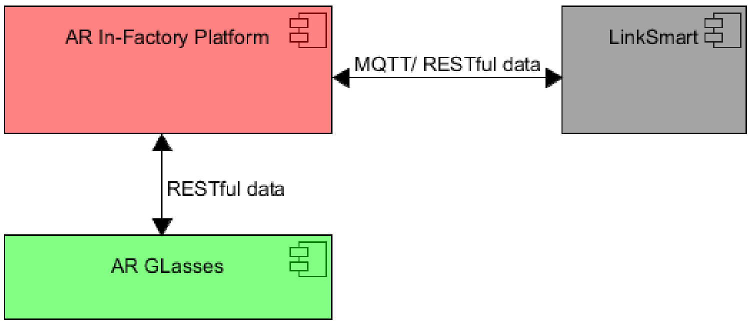


Figure 19 - AR Glasses integration with AR In-Factory Platform and LinkSmart

Currently, LinkSmart integration is not implemented; this will be presented during the second half of the project within the second iteration of this deliverable. In particular, it will create a Device Manager for the AR Glasses able to send all the required information retrieved by LinkSmart directly to them, as well as to send the information gathered from the AR Glasses to the LinkSmart.

3.7 *AR In-Factory Platform*

AR In-Factory Platform component is an app for mobile devices created with the purpose of reproducing the Augmented Reality enriched Standard Operative Procedures (SOP) created with the Creation Tool component. It shows to the operator the instructions present in the SOP, the multimedia contents attached (images, videos, audio clips, 3d models/animations, digital documents) and creates an Augmented Reality view of the workplace, showing 3d models and animations on top of the device's camera view.

The AR In-Factory Platform is able to automatically evaluate conditional relations. Conditional relations are created in the Creation Tool component and are a way to condition the instructions' order. Any instruction's execution can be tied to the evaluation of a condition based on LinkSmart devices' values. AR In-Factory Platform automatically evaluates conditional relations being able to connect to LinkSmart and to retrieve values from the devices connected to it.

LinkSmart integration has been achieved using a plug-in based solution. A plug-in component has been developed to connect to LinkSmart Resource Catalog and Service Catalog, to read Devices' and Services' configurations from them and to get values from Devices and Services using the interfaces they expose.

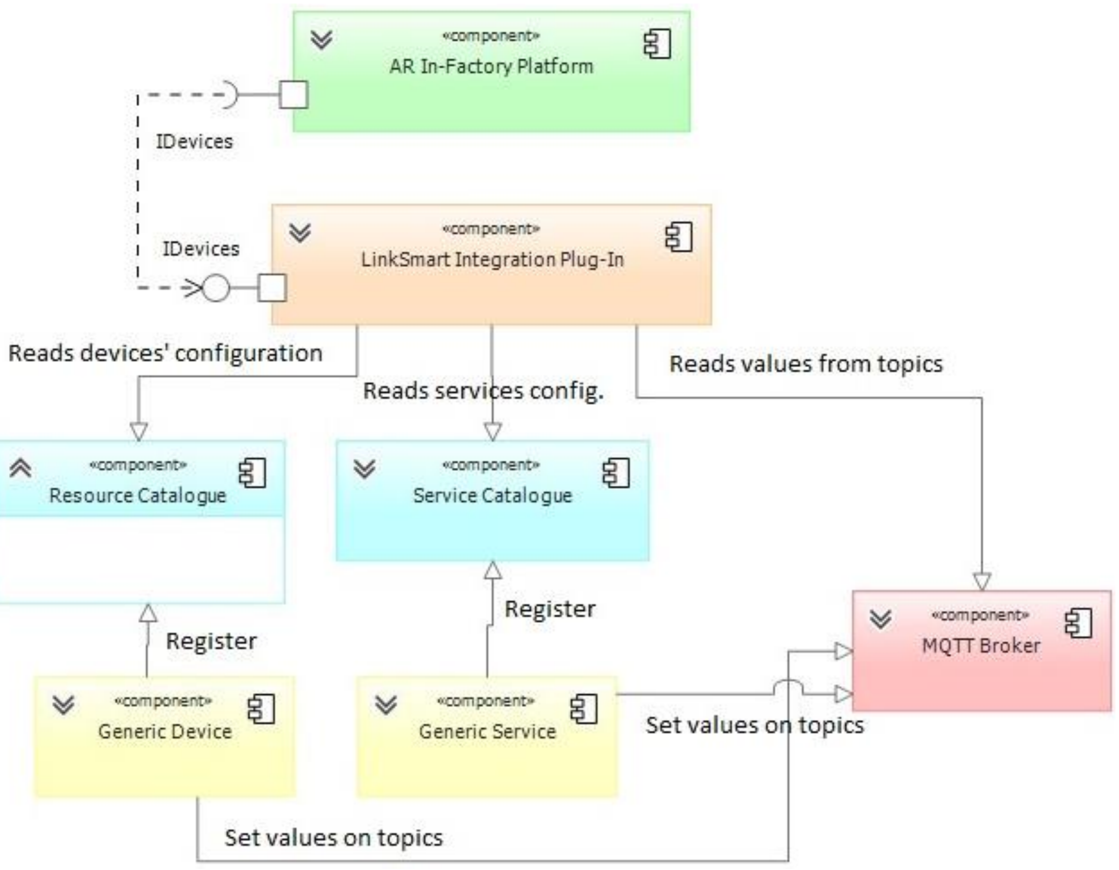


Figure 20 - AR In-Factory Platform integration with LinkSmart

Currently the AR In-Factory Platform's integration with LinkSmart has been successfully tested in a test environment. The integration has been tested with two different LinkSmart instances: one hosted by ITI and one hosted by CERTH.

The AR In-Factory Platform is ready to be deployed at CPERI to test it in a real case scenario. Following tests' result minor changes and fixes will be done if required.

To sum up, Table 2 presents the current status of integration of the SSN components and the AR In-Factory Platform in the three shop floors (COMAU, CPERI, SUNLIGHT).

Table 2 - Status of integration of SSN into LinkSmart

SatisFactory component	Responsible Partner	Integration With Resource Catalog	Integration with Service Catalogu	Deployed in CPERI	Deployed in SUNLIGHT	Deployed in COMAU	Notes
UWB Devices	**ISMB**	Yes	Not-Applicable	Yes	No	No	Integrated in **CPERI** in May 2016
Robust Wireless Temperature Sensor	**ISMB**	Yes	Not-Applicable	Yes	No	No	Integrated in **CPERI** in May 2016
Comfort Level Measurement Sensor	**CERTH**	Yes	Not-Applicable	Yes	No	No	A sensor has been integrated in **CPERI** in July 2016
Automation System - Legacy Sensors through the **PDEC**	**CERTH**	Yes	Not-Applicable	Yes	Not-Applicable	Not-Applicable	Data through PDEC are sent to LinkSmart Broker through MQTT (April 2016)
Depth and Thermal Cameras	**CERTH**	Yes	Not-Applicable	Yes	No	No	None
AR Glasses	**GlassUp**	Not-Applicable	Not-Applicable	Yes	No	No	Direct Integration with resource catalog and service catalog not expected
AR In-Factory Platform	**REGOLA**	Yes	Yes	Yes	No	No	Component has been deployed but still not configured to interact with the others LinkSmart devices/services

4 CONCLUSIONS

This document presents recent developments in LinkSmart middleware that is used in SatisFactory platform for device integration, event notification and inter-service communication. A new approach to architecture and improved design based on the deployment scenarios is presented. The concepts and models of the current architecture, which provides unified abstractions for simplified device integration, are thoroughly explained. The core services of the middleware and their APIs are also briefly explained. An event aggregation component has been developed for storing event notifications from device and services into CIDEM repository service. Furthermore, an introduction to available Smart Sensor Networks is given; corresponding Device Managers and their integration with middleware for event notification and collaboration are briefly explained. The deployment status of different Device Managers onto the pilot sites is also given.

The technical descriptions and specifications provided in this document are subject to change during the course of the project. For example, some minor changes in the Resource and Service Catalogs APIs are envisioned. An authentication and authorization layer along with persistence backend will also be added. The integration of social collaboration environments is also a subject of next iteration. Therefore changes in middleware services, integration and deployment status of all relevant SatisFactory components onto the pilot sites will be reflected in a final M30 deliverable.

REFERENCES

[1] Al-Akkad, A., Pramudianto, F., Jahn, M., Zimmermann, A. (2009): “Middleware for building pervasive systems”. International Association for Development of the Information Society (IADIS): International Conference Applied Computing, Rome, pages 1–8

[2] Eikerling, H., Gräfe, G., Röhr, F., Schneider, W. (2009), “Ambient Healthcare- Using the Hydra Embedded Middleware for Implementing an Ambient Disease Management System”. In Proceedings of the Second International Conference on Health Informatics, Portugal, pages 82–89.

[3] Jahn, M., Jentsch, M., Prause, C. R., Pramudianto, F., Al-Akkad, A., Reiners, R. (2010). „The Energy Aware Smart Home” In 2010 5th International Conference on Future Information Technology, pages 1–8

[4] Reiners, R., Zimmermann, A., Jentsch, M., Zhang, Y. (2009). Atomizing home environments and supervising patients at home with the hydra middleware: application scenarios using the hydra middleware for embedded systems“. CASTA '09 - Proceedings of the first international workshop on Context-aware software technology and applications, pages 9-12

[5] M. Compton, P. Barnaghi, L. Bermudez, R. García-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W. D. Kelsey, D. Le Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, and K. Taylor, “The SSN ontology of the W3C semantic sensor network incubator group,” Web Semant. Sci. Serv. Agents World Wide Web, vol. 17, pp. 25–32, Dec. 2012

[6] A. Gyrard, S. K. Datta, C. Bonnet, and K. Boudaoud, “Standardizing Generic Cross-Domain Applications in Internet of Things”

[7] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” 2000

[8] “JSON-LD 1.0. A JSON-based Serialization for Linked Data”, W3C

[9] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, “The Many Faces of Publish/Subscribe” ACM Comput Surv, vol. 35, no. 2, pp. 114–131, Jun. 2003

[10] OASIS, “MQTT Version 3.1.1.” 2014

[11] SatisFacory consortium, “Satisfactory System Architecture”, delivered at M16, April 2016

[12] SatisFactory consortium, “Development of Intelligent IoT Infrastructure for the Smart Sensor Network”, to be delivered at M20, August 2016

[13] SatisFactory consortium, “Context-Aware Incident Detection Engine”, to be delivered at M20, August 2016

[14] SatisFactory consortium, “Industrial lab use case Set-up and Demonstration”, delivered at M16, April 2016

ANNEX – Resource Description Files for components deployed at CPERI Pilot Site

1. Resource Description File for UWB-GW

```
{ "id": "satisfactory/GW1",
  "type": "Device",
  "name": "GW1",
  "meta": { "id": "01",
           "ip_address": " 160.40.2.208 ",
           "site_id":"CPERI",
           "broker": " 160.40.2.206 ",
            "port": 1883},
 "description": "uwb gateway device",
 "ttl": 120,
 "resources": [{ "id": "ismb/uwb/tag1",
              "type": "Resource",
              "name": "100102",
              "meta": {},
              "protocols": [{ "type": "MQTT",
                           "endpoint": {
                           "broker": "tcp:// 160.40.2.206:1883",
                           "topic": ["/satisfactory/GW1/ismb/uwb/tag1/position",
                                  "/satisfactory/GW1/ismb/uwb/tag1/alert"]},
                           "methods": ["PUB"],
                           "content-types": []}],
              "representation": {}
            }]
}
```

2. Resource Description File for RWTS-GW

```
{ "id": "satisfactory/WSN",
  "type": "Device",
  "name": "WSN",
  "meta": { "id": "01",
           "site_id":"ismb",
           "broker": "160.40.2.206",
           "port": 1883 },
 "description": "WSN gateway",
 "ttl": 120,
 "resources": [{ "id": "ismb/wsn/temperaturesensor1",
              "type": "Resource",
              "name": "temperaturesensor1",
              "meta":{"id":"ebb1:0:0:0:c30c:0:0:226"},
              "protocols": [{ "type": "MQTT",
                           "endpoint": {
                           "broker": "tcp://130.192.85.179:1883",
                           "topic":[
                           "/satisfactory/WSN/ismb/wsn/temperaturesensor1/temperature",
                           "/satisfactory/WSN/ismb/wsn/temperaturesensor1/channelinfo",
                           "/satisfactory/WSN/ismb/wsn/temperaturesensor1/error"]},
                           "methods": ["PUB"],
                           "content-types": []}],
               "representation": {}
             }]
}
```

3. Resource Description File for Depth Cameras

```
{ "id": "satisfactory/DepthCameras",
  "type": "Device",
  "name": "DepthCameras",
  "meta": {},
  "description": "CPERI sensor",
  "ttl": -1,
  "resources": [{ "id": "satisfactory/DepthCameras/Stream1",
              "type": "Resource",
              "name": "Stream1",
              "meta": { “store_in_cidem”: “true”},
              "protocols": [{ "type": "MQTT",
                             "endpoint": { "url": "tcp://160.40.2.206:1883",
                                          "topic": "/satisfactory/incidents"
                                        },
                             "methods": ["PUB"],
                             "content-types": []
                           }],
              "representation": {},
              “device”: “satisfactory/DepthCameras”
              }]
}
```

4. Resource Description File for Thermal Cameras

```
{ "id": "satisfactory/ThermalCamera1",
  "type": "Device",
  "name": "ThermalCamera1",
  "meta": {},
  "description": "CPERI sensor",
  "ttl": -1,
  "resources": [{ "id": "satisfactory/ThermalCamera1/Stream1",
              "type": "Resource",
              "name": "Stream1",
              "meta": { "store_in_cidem": "true"},
              "protocols": [{ "type": "MQTT",
                             "endpoint": { "url": "tcp://160.40.2.206:1883",
                                          "topic": "/satisfactory/incidents"
                                        },
                             "methods": ["PUB"],
                             "content-types": []
                           }],
              "representation": {},
              "device": "satisfactory/ThermalCamera1"
              }]
}
```