Boosting DR through increased communIty-level consumer engaGement by combining Data-driven and blockcHain technology Tools with social science approaches and multi-value service design

Deliverable D2.4 Cross-domain Data & Service Interoperability

Authors: Andrej Čampa (COM), Klemen Bregar (COM), Vjekoslav Delimar (ISKRA), Denisa Ziu (ENG), Giannis Kazdaridis (DOMX), Mente J. Konsman (TNO), Jevgenia Kask Savchenko (SUN)

Imprint

Title:	**Cross-domain Data & Service Interoperability**
Contractual Date of Delivery to the EC:	31.10.2021
Actual Date of Delivery to the EC:	29.10.2021
Authors:	Andrej Čampa (COM), Klemen Bregar (COM), Vjekoslav Delimar (ISKRA), Denisa Ziu (ENG), Giannis Kazdaridis (DOMX), Mente J. Konsman (TNO), Jevgenia Kask Savchenko (SUN)
Participant(s):	COM, ENG, ISKRA, TNO, DOMX, SUN
Project:	Boosting DR through increased community-level consumer engagement by combining Data-driven and blockchain technology Tools with social science approaches and multi-value service design (BRIGHT)
Work Package:	WP2 – BRIGHT Technology and Novel Multi-Value Service Design
Task:	T2.3 – Data Models & Service and Platform Interoperability Specifications
Confidentiality:	Public
Version:	1.0.0

Consortium - List of partners

Table 1. Consortium partner list

Partner no.	Short name	Name	Country
1	ENG	ENGINEERING - INGEGNERIA INFORMATICA SPA	Italy
2	TUC	UNIVERSITATEA TEHNICA CLUJ-NAPOCA	Romania
3	IMEC	INTERUNIVERSITAIR MICRO-ELECTRONICA CENTRUM	Belgium
4	COM	COMSENSUS, KOMUNIKACIJE IN SENZORIKA, DOO	Slovenia
5	SONCE	SONCE energija d. o. o.	Slovenia
6	ISKRA	ISKRAEMECO, MERJENJE IN UPRAVLJANJE ENERGIJE, D.D.	Slovenia
7	EMOT	EMOTION SRL	Italy
8	TNO	NEDERLANDSE ORGANISATIE VOOR TOEGEPAST NATUURWETENSCHAPPELIJK ONDERZOEK	Netherlands
9	CENTRICA	CENTRICA BUSINESS SOLUTIONS BELGIUM	Belgium
10	ASM	ASM TERNI SPA	Italy
11	DuCoop	DUCOOP	Belgium
12	CEL	CYBERETHICS LAB SRLS	Italy
13	DOMX	DOMX IDIOTIKI KEFALAIOUCHIKI ETAIREIA	Greece
14	APC	Asociatia Pro Consumatori	Romania
15	WVT	WATT AND VOLT ANONIMI ETAIRIA EKMETALLEYSIS ENALLAKTIKON MORFON ENERGEIAS	Greece
16	SUN	SunContract OÜ	Estonia

Table of Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

Table 2. List of Acronyms and Abbreviations

AMI	Automatic Metering Infrastructure
API	Application Programming Interface
BEMS	Building Energy Management System
BRIGHT	Boosting DR through increased community-level consumer engagement by combining Data-driven and blockchain technology Tools with social science approaches and multi-value service design
CEM	Customer Energy Manager
CHP	Combined Heat and Power
CIP	Consumer Information Push
CO2	Carbon dioxide
CSV	Comma-Separated Values
DB	Database
DHW	Domestic Hot Water
DR	Demand Response
DSO	Distribution System Operators
DT	Digital Twin
EC	European Commission
EMS	Energy Management System
EU	European Union
EV	Electric Vehicle
ESB	Enterprise Service Bus
ETSI	European Telecommunications Standards Institute
GDPR	General Data Protection Regulation
GW	Gateway
HBES	Home and Building Electronic System
HDLC	High-level Data Link Control
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation, and Air Conditioning
ID	Identifier
IED	Intelligent Electronic Device
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
LDN	logical device name
MSA	MicroService Architecture
PV	Photovoltaic
REST	Representational State Transfer
SAREF	Smart Applications REFerence
SASS	Singe Application Smart System
SM	Smart Meter
SOA	Service-Oriented Architecture
SoC	State of Charge

SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URL	Uniform Resource Locator
Wi-Fi	Wireless Fidelity
WP	Work Package
XML	Extensible Markup Language

Executive Summary

The objective of this deliverable is to define a framework for cross-domain data and service interoperability to be used in the BRIGHT consortium. Interoperability enables different services and devices to freely exchange information throughout all the domains presented in the BRIGHT services and pilots. This limits the need to rewrite all the existing code to be compatible with all different existing device and service APIs but instead it needs to be adapted only to comply with the BRIGHT cross-domain interoperability API. The aim is to make the data exchange as easy as possible and facilitate the development of data driven and cross-domain services.

To meet the interoperability objectives, the document covers various topics related to the definition of the architecture of the BRIGHT interoperability layer (see Figure 1). In addition, the overview of most relevant standards for achieving interoperability inside the BRIGHT project is presented and discussed. The BRIGHT consortium strives to reuse the technologies and standards available, therefore increasing the openness of the solution and enabling faster adaptation of services created in the case the solutions would be used and deployed outside the BRIGHT ecosystem.

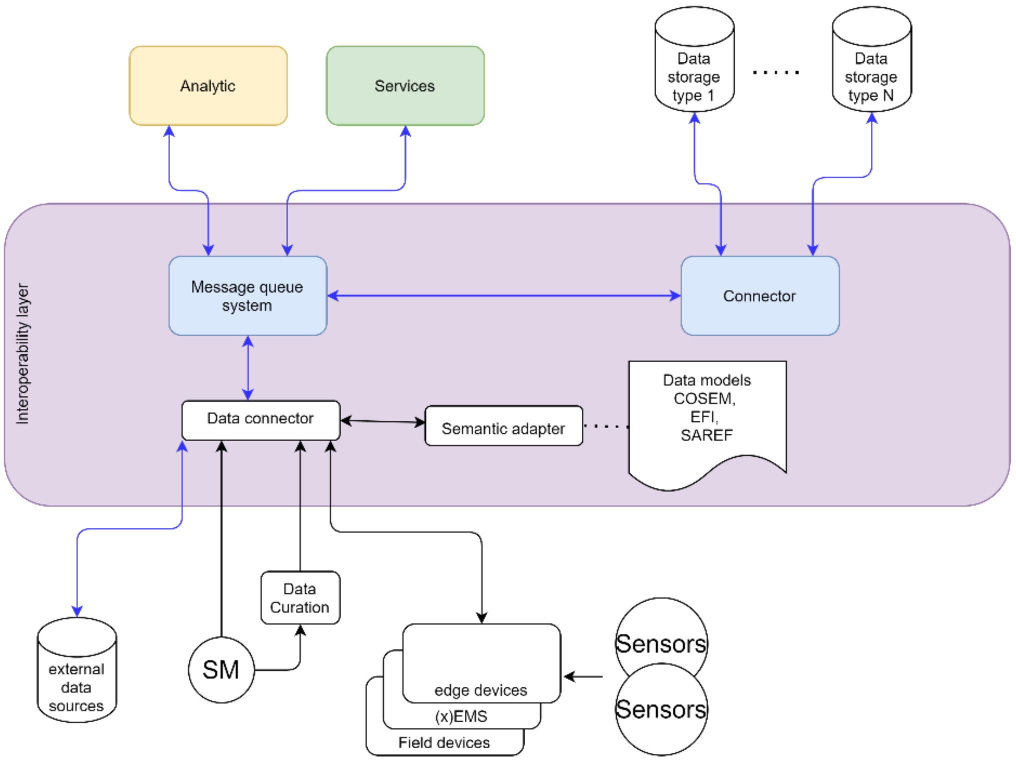


Figure 1. The architecture of the BRIGHT Interoperability layer

Following the definition and collection of the requirements for the interoperability layer, we have started to analyse data models theory and application of data models to ensure the BRIGHT ecosystem and its components can communicate in the same language by means of data exchange. The data model section is further expanded by the presentation of a harmonized list of already available data sources that will be used throughout the BRIGHT system and pilots. The list will be further extended if new information objects will be identified during the project lifetime.

Further into the document, standardised data exchange systems and ontologies for IoT and energy systems are presented (SAREF, S2 and DLMS/COSEM). The description of ontologies is followed by

the definition of APIs in the BRIGHT project (REST API, and Apache Kafka streaming API). It is later followed by the introduction of additional communication protocols that are used to access and report device information like MQTT protocol for lightweight publish-subscribe communication and unidirectional CIP protocol for access to the data from the metering device by the consumer.

The first analysis of the BRIGHT pilots, data sources and requirements taken by the T2.3 and presented by this deliverable is the basis for creating BRIGHT-specific ontology that will follow in the next version of this deliverable D2.5 scheduled for M19.

1 Introduction

Deliverable D2.4 aims to provide an overview of the BRIGHT Consortium's approach to interoperability, primarily to standardize cross-domain access to data and to unify service interoperability. The main components in this report are the first two bullet points, while the third point is still in progress and will be part of the second version of this deliverable D2.5 scheduled for M19:

- Overview of most relevant standards related to BRIGHT project interoperability. The standards are grouped according to the scope (data, service), the level at which they are used and the type of data.

- Preliminary data models for interfacing with BRIGHT sources and services once developed and deployed.

- A multi-purpose vocabulary, that covers not only technical data, but also data generated and obtained from social-science driven insight.

Once the design is finalized (i.e., *D2.3 DR technologies and tools specification* in M12) and service development begins, the specified components will be reviewed and published in the second version of this deliverable to address additional requirements found by a wider consortium and potentially propose a more efficient solution that will be better aligned to the partner's technical needs.

1.1 Scope of the document

The presented deliverable describes the methodology for unifying the approaches taken among different stakeholders and is a continuation of the work done in document D2.1 *User group needs, req. & advanced DR engagement scenarios* towards interoperability. The aim of the document is to identify different standards and analyse them according to the needs of pilots and service developers. Once analysed, the document proposes the solution aligned inside the BRIGHT project and the requirements defined by EC to unify the processes across the EU. The outputs of the document are well-defined approaches with examples across different spectra mainly:

- Overview of existing technologies
- Creation of BRIGHT specific interoperability solution
- Analysis of different data models, from low-level data models used at sources and high-level data models.
- Gathering of information about available and planned information objects and harmonizing them.
- Preparing harmonized data for the creation of a BRIGHT data model solution.

The logical structure of the document is therefore

- *Section 1* provides an overview of the document and existing solutions for achieving interoperability. It also defines the BRIGHT interoperability layer.

- *Section 2* addresses data models and harmonization of data sources to be used in the design of the BRIGHT data model.

- *Section 3* gives an overview of the data exchange system and ontologies. It focuses on the two main ontologies SAREF and S2 Communication. At the same time, it also emphasises using SM low-level data model DLMS/COSEM, since SMs are becoming one of the most important data sources in DR solutions and built-in data models can also leverage interoperability at the edge site.

- *Section 4* analyses key APIs for creating the BRIGHT interoperability solution

- *Section 5* analyses the protocols relevant to BRIGHT pilots.

The document ends with the conclusions and the questionnaire on the information objects available in each pilot, which form the basis for creating a BRIGHT data model.

This document uses the BRIGHT architecture defined in D2.3 and analyses, creates and positions the interoperability layer in the architecture to be used by all partners to establish interoperability. The outputs of the document are the requirements and architecture of the interoperability layer that needs to be taken into consideration when setting up pilots (WP7) and when developing BRIGHT solutions in WP4, WP5 and WP6.

1.2 Cross-domain data interoperability

Data interoperability refers to the system's capacity to map different data sources to the data model. Such a system can freely create, consume and exchange data with a clear knowledge of its context and meaning. In order to reach data interoperability from the data source, we have to follow a few crucial steps, as shown in Figure 2. The main components of the flow are:

- *Data source:* is any source that generates data, this could be simple devices (senors, IED, IoT, xEMS, etc.), databases or even non-technical data acquisition obtained with various methods. For example, describing socio-economic aspects with analytical methods or manual direct or indirect acquisition of data from actors using various questionaries (WP3).

- *Data Acquisition:* This is the step that is in charge of integrating heterogeneous data sources and facilitating its integration. The data acquisition component has to deal with so-called 3Vs volume, velocity and variety [1], which can be extended to 7Vs [2], but this is out of the scope of this delivery. Furthermore, the data can be structured or unstructured, it can come from various sources with limited connectivity.

- *Data Curation:* The next step during the typical data flow is the process that can be done on the raw data or on the big data. It is responsible for pre-processing the data such as cleansing, anonymisation and semantic enrichment.

- *Data Storage:* Different types of data storage are needed; therefore, most of them must be considered to reach full interoperability. In most cases, data storage is done with the databases. However, in case of getting closer to the source of the data, even other types of data storage might be available that mostly deal with a small chunk of data in real-time (e.g. temporary such as registers and memory or persistent like files). In the case of a common data lake or distributed storage is important for achieving autonomous, real-time or batch processing. Therefore, storage should cover not only one type of database but many different types of databases such as relational, non-relational databases and time-series databases.

- *Data Security:* In parallel to all the above described components, the security of the data is of paramount importance. Data protection needs to be assured through all the levels, furthermore, anonymization with all possible ethical issues needs to be considered. Data security is more in detail described in *D2.2: Privacy, Ethics and Legal Requirements.*

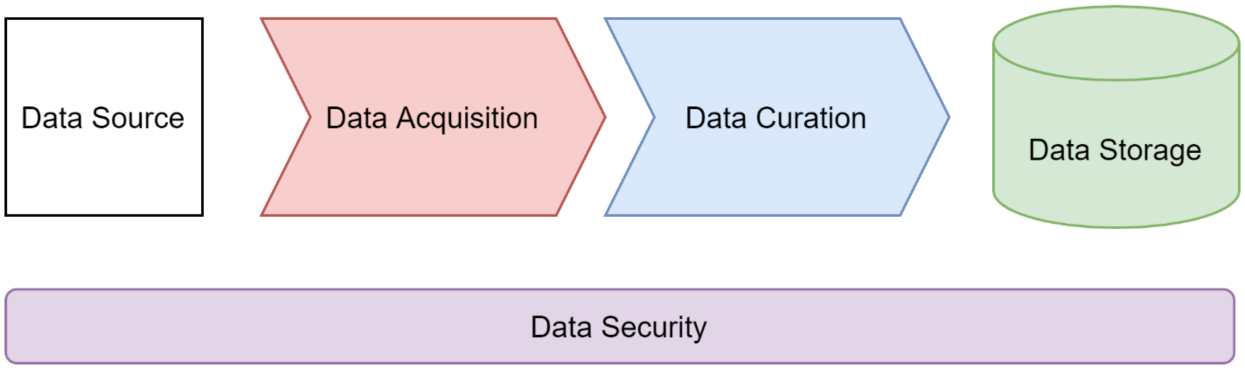


Figure 2. Typical Data flow

In Figure 3, the data layer stack is presented. The data from the source (e.g. sensor) can transform and evolve when it passes different processing layers (Edge, Fog and Cloud). To satisfy the speed and reach interoperability, the data will gradually evolve from raw data to big data that is semantically enriched with selected ontologies during the process of acquisition and curation. First, at the Edge, raw data from sensors and other producers usually get processed and temporarily stored. Data at the Edge can be used for real-time processing and creating real-time loops. The processed data can be pushed to Fog or Cloud for additional processing to establish higher observability and better awareness of the system as a whole. The Fog layer extends the Edge layer to the Cloud, but it still ensures decentralized data processing. This ensures a better awareness of the overall system compared to the Edge layer while creating an additional intelligent layer that extends cloud computing capabilities closer to the Edge. In the case of Edge and Fog computing, sensitive data can remain inside the network and only non-sensitive data is pushed to the Cloud for central processing. Cloud fulfills the need of accessing big data quickly and ensures easy scalability.

Since we are dealing with data models and ontologies, we first distinguish between these two. Ontology provides a comprehensive hierarchical view of a domain and aims to develop general taxonomies of what exists. On the other hand, the data model provides a flat and partial representation of the data and aims to develop classifications within a particular application domain [3]. More detailed features are summarized in Table 3.

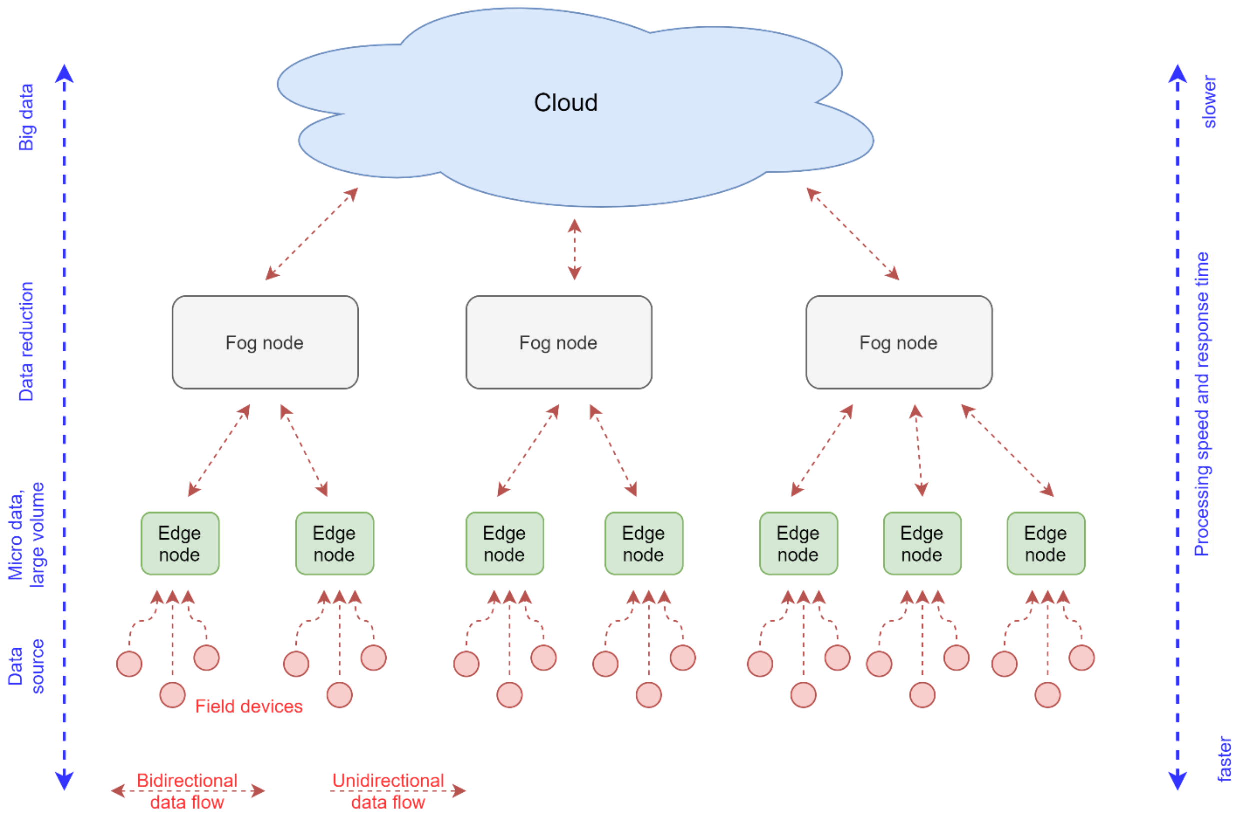


Figure 3. Relationship between different processing layers.

Table 3. Data modelling and Ontology main properties [4]

Data Modeling	Ontology
Partial account of conceptualization	
Depends on the specific task and needs	Application independent
Informal agreement between developer and user	Generic knowledge
Not intended to be shared	Sharable and reusable
Structure and integrity	Semantics
Application-specific	Open environment

The BRIGHT consortium will strive to reuse existing schemas, by determining the domain and scope of the models, defining common entities and entity hierarchy, defining common attributes and classes, and lastly mapping each pilot data to a common data model.

In the case of ontology, we strive to reuse existing ontology and, if needed, extend it with the parts found during the process of analyzing existing ontologies and pilot existing data sets and requirements.

1.2.1 Data sharing and leveraging

Additionally, to create added value from the data for the purposes of the internal stakeholders, it may be beneficial to share the data (raw or processed) also to third parties. Exploiting the data of the different users of various BRIGHT communities allows the creation of new added-value services that might be outside this project's scope. However, timely access to the cleaned data is crucial for relevant parties to achieve fair and healthy market competition that will be mainly beneficial for the energy market and end-users. In any case, GDPR compliance needs to be respected.

1.3 Service interoperability

The basis for service interoperability are well-defined data models. The BRIGHT project will follow already developed standards in service interoperabilities based on the distributed service-based architecture. On the one hand, microservice architecture (MSA) is an application architectural style and an application-scoped concept [5]. The main features of microservices are that they decouple the components and thus simplify and improve the following:

- *Agility*: New technology can be introduced without affecting the rest of the components. Each component can be independently tested.

- *Scalability*: Particular components can be scaled without affecting others achieving the fastest possible response to workload demand and more efficient use of resources.

- *Resilience*: Failure of one service won't affect others.

On the other hand, Service-oriented architecture (SOA) is an approach to develop various components that takes advantage of reusable software components and services via the service interface. SOA connects applications together, making it easier to share data and functionality [6]. The main advantages of SOA are:

- *Reusability*: Self-contained and loosely coupled components in SOA enables to reuse of these components without influencing other services.

- *Maintainability*: Each service is an independent unit which makes it easy to update and maintain.

- *Parallel development*: SOA consists of layers, it enables the development of services in parallel.

- *Orchestration* of services.

The more detailed conceptual architecture of MSA and SOA is shown in Figure 4 [7]. The main advantage of the evolution of the services to SOA is that it eliminates point-to-point integration that developers had to create for each project, exposing the parts of the components through SOA. It eliminates the need to recreate deep integration with a new project. Therefore each service consists of three components:

- *Interface*, that handles link between actor/user and service

- *Contract*, defines interactions between the service provider and service consumer. In essence, the contract describes each service and defines information exchange.

- *Implementation*, is the service code.

Enterprise service bus (ESB) in the SOA is the layer that translates a message to the correct type and sends translated message to the correct service. In the SOA with ESB any application can behave as a server or client, providing greater flexibility and agility to the communication. The concept of ESB [8] is analogous to the bus concept in computer hardware architecture, as shown in Figure 5. The main functions of an ESB are:

- Routing of messages
- Monitor and control of communicated messages
- Resolve communication disputes among services
- Control deployment and versioning of services
- Enforce proper quality of communication service.

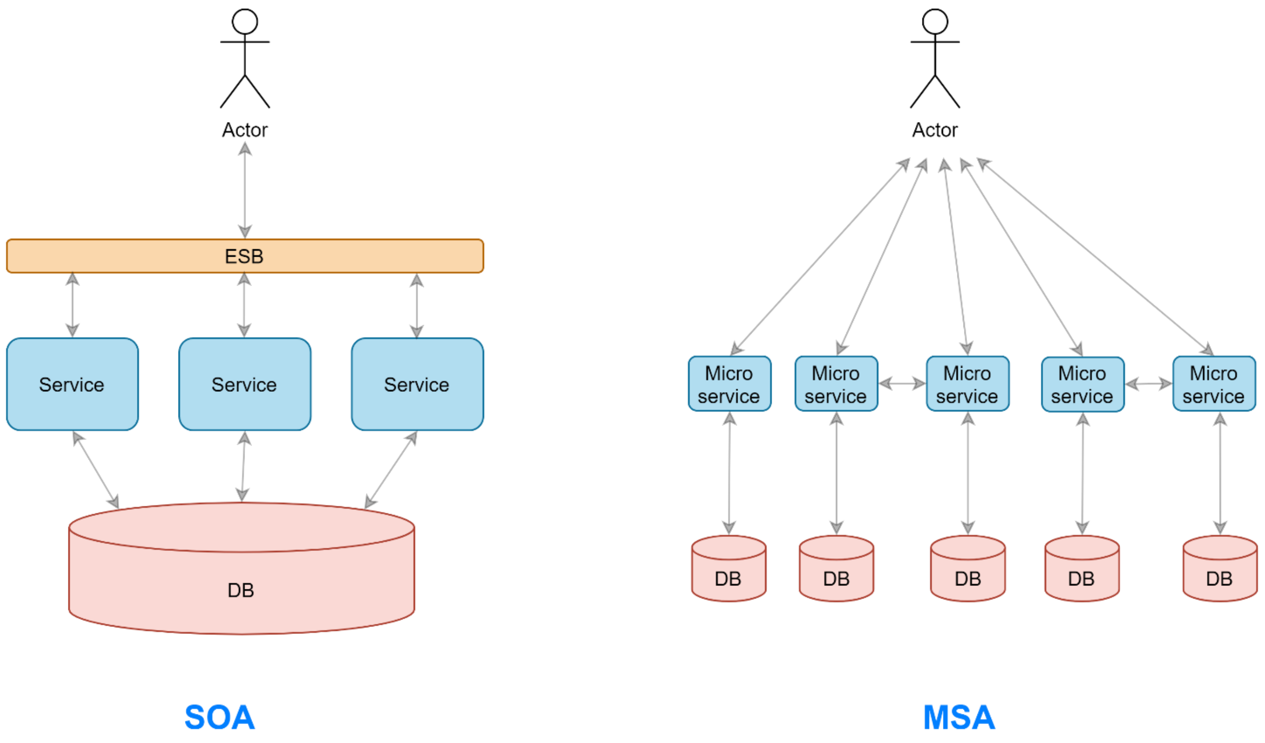


Figure 4. Service-based architectures MSA and SOA

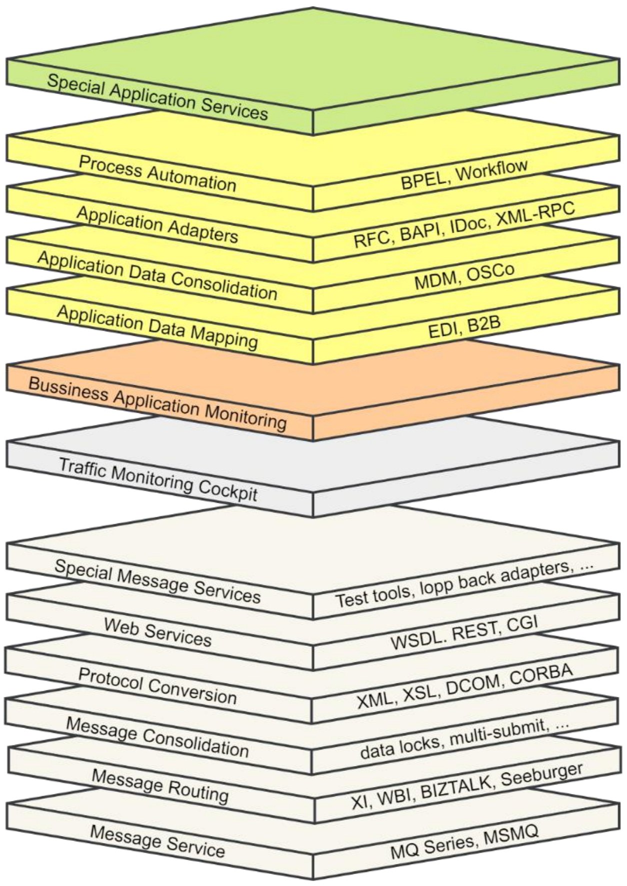


Figure 5. Example of ESB stack, as shown in ref. [8]

1.4 Interoperability requirements

In the BRIGHT project, four pilots are dealing with DR based on different communities, and therefore partners are developing different analytic tools for different use cases as is described more in detail in *D2.1 User group needs, req. & advanced DR engagement scenarios*. In each pilot, different partners are involved with their own datasets, platforms and tools. Some of the tools will be directly deployed at the edge, and some tools might be developed for one pilot and also tested at other pilots. Therefore, BRIGHT will not impose a single centralized platform that partners need to use. But instead, it will promote a federated platform, consisting of different platforms and components from the partners that are able to exchange data and services with each other. For connecting different platforms an interoperability layer is introduced with the following layers [9,10]:

- *Organizational*. The most complex and high-level interoperability layer, where each request must obey a master plan (business process, workflows, etc.). This level usually requires human to human interaction.

- *Semantic.* This level deals with the meaning of the data, it improves the understanding of the data that might be lost in low levels sharing of data. The common semantics need to be agreed beforehand, this means compatibility in ontology, rules and workflows.

- *Syntactic.* Heterogenous systems such as systems using high-level programming languages need to talk to each other. Therefore this category deals with the form that needs to be defined to understand the format and structure of the data. An example of this category is REST API, where the message's contents are serialized to be sent over the channel. The data format used can be XML, JSON etc., which provides syntactic interoperability so systems in different programming languages can exchange data with each other.

- *Connectivity.* The interoperating system must support the exchange of data at different levels starting from the lowest hardware or machine level. Today at the machine level, interoperability is standardized and implemented in a compiler, virtual machine and operating system. On a higher level, the interoperability is assured with enclosing content of information with message control information, for example an HTTP request which is sent over a network using popular TCP/IP protocol that hides interaction details from the user.

The interoperable system must follow these levels. Agreement on more levels means higher interoperability. In BRIGHT, we will focus mostly on Connectivity, Syntactic and Semantic interoperability. The interoperability seen from a different perspective for IoT example is shown in Figure 6 [11].

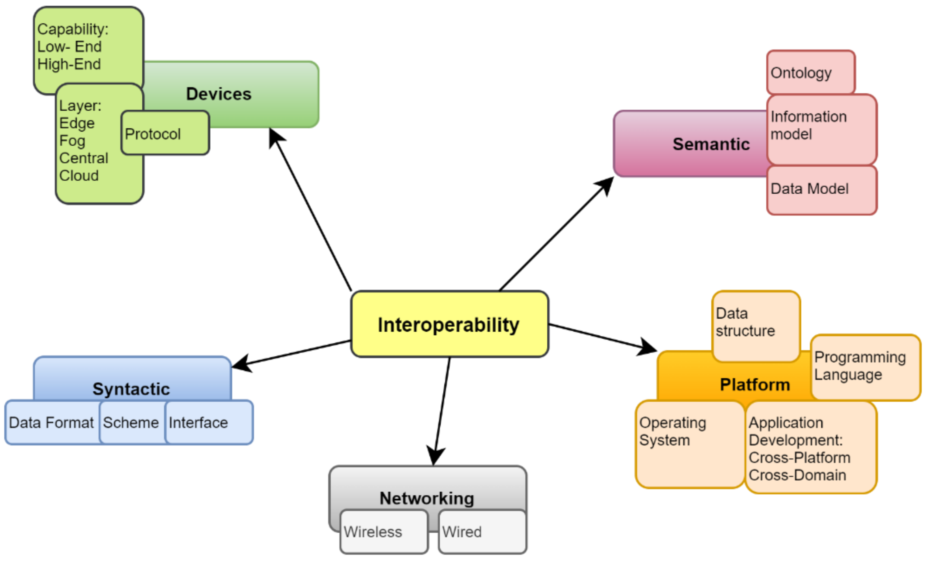


Figure 6. Example of IoT taxonomy

This interoperability layer has to meet the following requirements:

- It must define a set of common data models, for example, semantic data models such as ontologies.

- Since BRIGHT develops services for interdisciplinary citizen engagement, the existing ontologies will be reused and expanded.

- A developed set of common data models must be open for later reuse and to enhance its adoption.

- Fusion of heterogeneous data that will be used (e.g. Files (csv, xml, JSON, etc.), logs, relational and non-relation databases).

- Define common API to improve integration among different platforms (it is beneficial to reuse the existing solution).

- At the device level, a secure and authorized connection is required.

- At the communication level, sensitive information might be exchanged via encrypted channels.

- The user level, role and access permission must be defined so only users with permitted functionalities can use the system.

- Privacy and use of personal data must comply with GDPR

Additionally, the interoperability layer needs to be designed in a way it meets the requirements related to Privacy, Cybersecurity, Ethics and Legal dimensions that are in more detail described in *D2.2: Privacy, Ethics and Legal Requirements*.

Security is another important aspect of the interoperability layer to preserve data privacy on all levels. Encrypted communications are used in all interactions between components of the BRIGHT platform that involves data exchange. In practice, this means to encrypt data in transit between applications (e.g. using Transport Layer Security (TLS) or Secure Socket Layer (SSL) protocols) and enable authentication (e.g. using digital certificates X.509).

1.5 The architecture of the interoperability layer

Taking into account the functionalities and requirements defined by the BRIGHT project, we have defined the architecture of the interoperability layer with the components depicted in Figure 7. In the figure, for the sake of simplicity, only the main logical components and interactions are depicted since the final implementation can be slightly different in Pilots. The interoperability layer is responsible for collecting and managing data, semantic adaption or mapping and data integration.

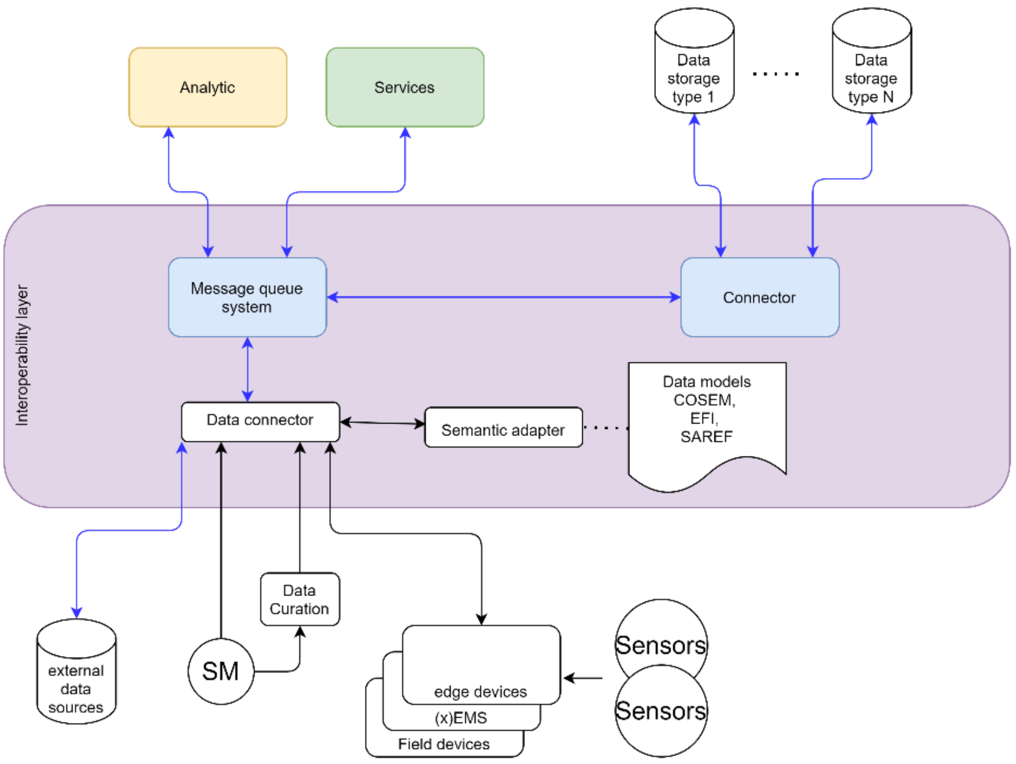


Figure 7. The architecture of the BRIGHT Interoperability layer with the components

1.5.1 Architecture components description

In the following section, the components of the interoperability layer are defined and described.

Component	**Data connector**
Description	This component provides a necessary connection to field devices that can not meet the requirements for interoperability and can not be modified due to various restrictions. Usually, the systems connected to this connector are Legacy devices already available in the Pilots and Proprietary systems.
Main functionalities	• Read data from legacy and proprietary systems • Read from API • Read CSV, JSON files • Read from DB • Push and pull enriched data
Interactions with other components	• SM: supporting legacy and new smart meters • Edge devices: Such as EMSs and GWs and others that do not support the integration of semantic adapter • Semantic adapter: to enrich the raw data into semantic data using common data models • Message queue system: to send the updated data enriched with the common data models

Component	Semantic adapter
Description	This component performs the data transformation from raw data or non-semantic data to semantic data. The component receives the data and passes the enriched data to upstream components. This component should be able to work on streams and batch data, which is defined by the connector that the component connects to.
Main functionalities	• Data transformation (raw to semantic, or align with BRIGHT ontologies) • Can work with real-time and batch data • Rules for each source of the data (connector, source, type, …)
Interactions with other components	• Data connector: to enrich the raw data obtained from the data connector into semantic data using common data models

Component	Message queue system
Description	This component expands the ordinary point-to-point messaging system to distributed publish-subscribing messaging system. The queuing enables to divide the processing of data over multiple instances to increase scalability. In addition, fault tolerance is ensured by storing data on a disk, configured by retention policies.
	•
Main functionalities	• Publish and subscribe to streams • Fault-tolerant (replication) • Distributed system
Interactions with other components	• Data connector: to obtain the enriched data from legacy systems • Connector: to expand the functionalities on data streams with data processing and storage

Component	Connector
Description	This component guarantees a scalable and reliable stream of data between the Message queue system and other components. In addition, it is a centralized data hub enabling easy integration between databases, file systems, etc.
Main functionalities	• Enables quick definition of connectors to move large data sets in and out from queue system • Connect to various databases, file systems and dashboards to make them available for stream processing • Support real-time and batch-oriented systems

Interactions with other components	• Message queue system: to connect to data streams

2 Data Models

A data model is a conceptual or visual tool to describe and represent the information about data elements and the connections between them [12] and deals with data types and structures, operation and consistency. It is meant to represent the real world, the information that can be used for later use and would generate added value. The data model focuses on what data is needed and how it should be organized. The primary goal of using data models are [13]:

- Ensures the data objects are accurately represented since omission might lead to incorrect results.
- Data model structure helps to define relations.
- Provides a clear picture of the data.
- Help to identify missing/redundant data.
- It unifies the part of the IT infrastructure to be more easily upgradable and easier to maintain.

2.1 Three Perspectives of a Data Model

Different types of data models exist depending on the current stage of the data modeling process [14]:

- *Conceptual or domain data model:* defines the high-level structures, concepts and semantics of a domain. It is usually created as a precursor to later stages of data model development.
- *Logical data model:* this model extends the conceptual data model by clarifying the various logical entities (types or classes), data attributes and their relationships.
- *Physical data model:* presents the actual data model implementation.

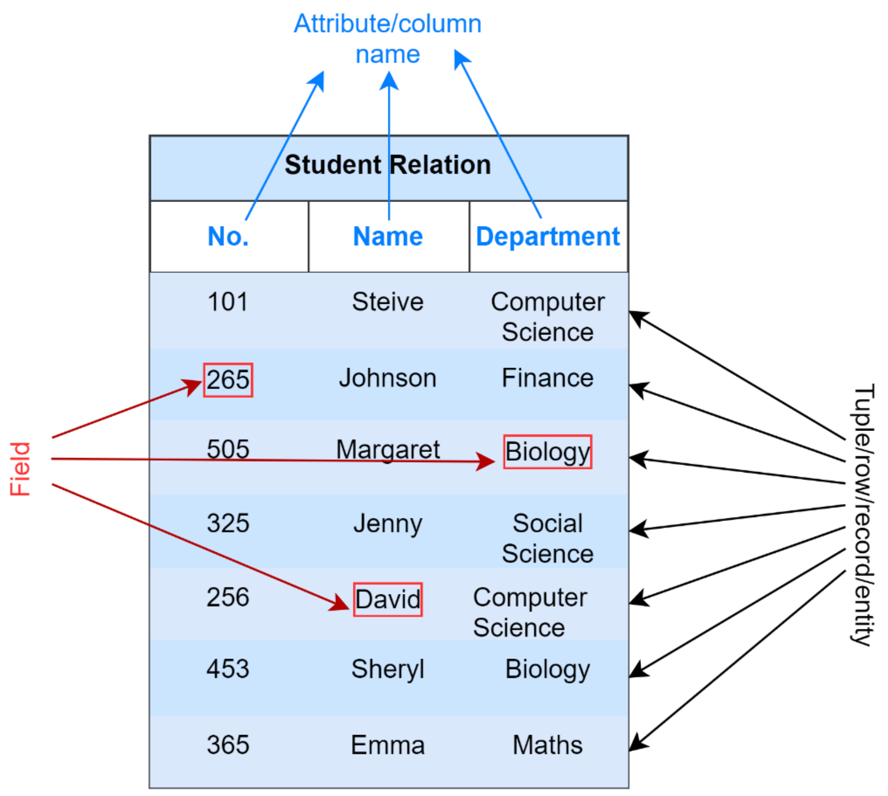
These three data model perspectives are relatively independent of each other. This property of intermodel independence enables the change in one data model type without affecting the other two data models (e.g. changing database storage technology etc.).

2.2 Types of Data Models

Several types of data models exist in the literature and, from them, mainly generic data models and semantic data models are being used.

Entity-Relationship Diagrams or as is frequently called entity-relationship model, is an abstract or conceptual presentation of structured data. It is produced by a relational schema modelling method called entity-relationship modelling to produce a conceptual data model type (semantic data model). It typically represents a model of a system and its requirements in a top-down fashion (created from abstract system representations created by the experts from the subject area and can be used as a template). They are mostly used in the information system design process during the requirements analysis to describe the system's information needs.

Generic Data model defines general relation types, for example, relation data model. A relation data model describes the data, its relationship, semantics and constraints on the data in the relational database [15] with an example shown in Figure 8.



Student Relation		
No.	Name	Department
101	Steive	Computer Science
265	Johnson	Finance
505	Margaret	Biology
325	Jenny	Social Science
256	David	Computer Science
453	Sheryl	Biology
365	Emma	Maths

Figure 8. Relation model with attributes, tuples and field

There are also other types of generic data models, such as database models, geographical data models, etc.

A semantic data model is a high-level description and structuring to improve the knowledge of variables, their interactions, priory, dependencies and correlations among them. The data is presented with named objects, sets of values, relationships and constraints over these objects. For example, an ontology is a schema that contains a set of:

- *Concepts*, to describe a certain reality (in our cases mostly devices or functions);
- *Relations* among concepts (type of appliance performs some function, e.g. HVAC performs a start or stop);
- *Axioms*, that present constraints of the concepts (e.g. the appliance should start in the time frame from 2 p.m. and 4 p.m.);

An overview of one of the ontologies is shown in Figure 10.
The data models define the common language for received and sent data among different components developed inside BRIGHT. This assures the rest of the components can understand the data according to the data model. Therefore, whenever there is an exchange of data among components from different platforms, the data needs to be exchanged according to common data models. However, for the internal communication among components, especially for systems with

limited processing capabilities at the edge, the more efficient data models can be used and might be more appropriate and may not be defined here.

According to the needs, we can define a few interaction scenarios:

- Services or components are developed from scratch. In this case, the easiest way is to build the components according to the common data models.

- Services or components developed from scratch, but they do not require semantic data. In this case, we do not enforce using the defined data models.

- Legacy tools that already use different models. In this case, a wrapper could be used that transform the data to the BRIGHT common data models.

2.3 Information objects

In the D2.3, the first identification of data coming from different sources were identified coming from different pilots. Since different services/tools or actors are involved in the project, the data harmonization needs to be performed with the following steps:

- Identify already existing data models that can be used as reference.
- Identify various data sources relevant to BRIGHT actors.
- Design data models.

In this document, only the first harmonization steps were performed, while the completely defined data models will follow in the later stage in D2.5.

2.3.1 Identify existing models

In the first version of the deliverable, we have identified and described the standards used by the BRIGHT project (sections 2.1, 2.2 and 2.3) and the BRIGHT data model will be built upon it.

2.3.2 Data sources

In this section, the initial list of data sources that need to be enriched and harmonized with ontologies are listed in Table 4. The final consolidated list will be available in D2.5. In the list, we have tried to include also the data sources that cover the social-related data to be gathered as a result of efforts in WP3. Since WP3 is still shaping its strategies for data collection, we have introduced two potenital objects as placeholders to be further considered in the second iteration of this document. The introduced social-related data objects are Schedule and Citizen input. Schedule does not relate to aparticular citizen, but to the typical representative of the group or its DT. In the case of Citizen input, beyond the actual data gathered through feedback tools such as questionnaires, the idea is to introduce the information object that enables the GDPR compliance from a technical point of view (e.g. the right to be forgotten). Indeed, all personal data gathered will be processed in compliance with existing rules and regulations, as detailed by the requirements in D2.2.

Table 4. List of identified objects and list of attributes

Object	Description	Attributes
Smart meter	Smart meter device, also advanced metering	- ID - Timestamp

	infrastructure (AMI) and any legacy electric energy meter	- Active Energy/power - Voltage - Current - Tariff
Energy meter	Any other metering device that is not a utility meter used by DSO	- Consumed Energy (day, night, total) - Current - Frequency - Voltage - Active power - Reactive power - Power factor - Timestamp
Energy meter heat	Heat generation, e.g. heat pump	- Operational state - Generated power - Input Temp. - Output Temp. - Energy - Active power - Timestamp
Generation	Generation of energy, such as PV.	- ID - Measurement Unit - Timestamp - Active Energy Import - Reactive energy - Current - Voltage - Frequency - Power - Total yield
Storage	Energy storage, mostly battery systems	- Timestamp - SoC - Charge/discharge Power - Operational state
Heat Storage	Accumulation of the heat in the water heat storage	- Temperature - Min. Temperature - Max. Temperature - Volume - Timestamp
EV	Electric vehicle	- ID - Timestamp - SoC
EV charging station	Electric vehicle charging station	- ID (station and socket) - Max power - Min power - Current

		- Voltage - Charging power - Price - Timestamp
Load	Flexible or any other types of loads (e.g. HVAC, heating devices)	- Consumed electric energy/power - Current - ID - Timestamp
Weather	Historical and forecasted weather for a location of interest	- Date - Time - Temperature - Precipitation - Humidity - Wind direction - Wind speed - Solar radiation - UV index
Calendar	Local/National calendar with all types of events (national holidays, other free days and bank holidays)	- Working days - Weekend - Public holidays and bank holidays
Property and measurements	e.g. indoor parameters (temperature, humidity, CO2)	- ID - Timestamp - Unit - Value
Smart Heating Controller	Remote monitoring of heating system parameters with the aid of smart heating controllers attached with the boilers of pilot users enable control and access of boiler's parameters	- Indoor/outdoor temperature, - Space heating temperature setpoint, - Domestic hot water temperature Setpoint, - Boiler water temperature, - DHW temperature, Heating usage, - Hot water usage, - Boiler modulation, - Gas consumption, - Timestamp
Private living unit	Mostly dwellings and office spaces	- Temperature - Temp. setpoint - District heating Temp. - District heat return Temp. - District heating flow - Tapping Temp.
BEMS	Building Energy Management system	- Zones Temperature - Humidity levels - Technical setpoints

		- Timestamp
Schedule	Scheduled activity for a particular user/group	- Activity - Day of the week - Start time - Stop time - Citizen group - User
Citizen input	An item holding info of user preferences settings	- User - Consent - Preference - Citizen group

2.3.3 Designing of the data models

The identified objects with their attributes are the base on which the BRIGHT project is starting to create the unified data model. To leverage the interoperability, the existing standards are going to be used where possible.

The data modelling process will be used to define and analyze the data requirements needed to support the processes in BRIGHT, Figure 9. Data models will provide a needed interoperability framework for data that will be used within the BRIGHT information system.

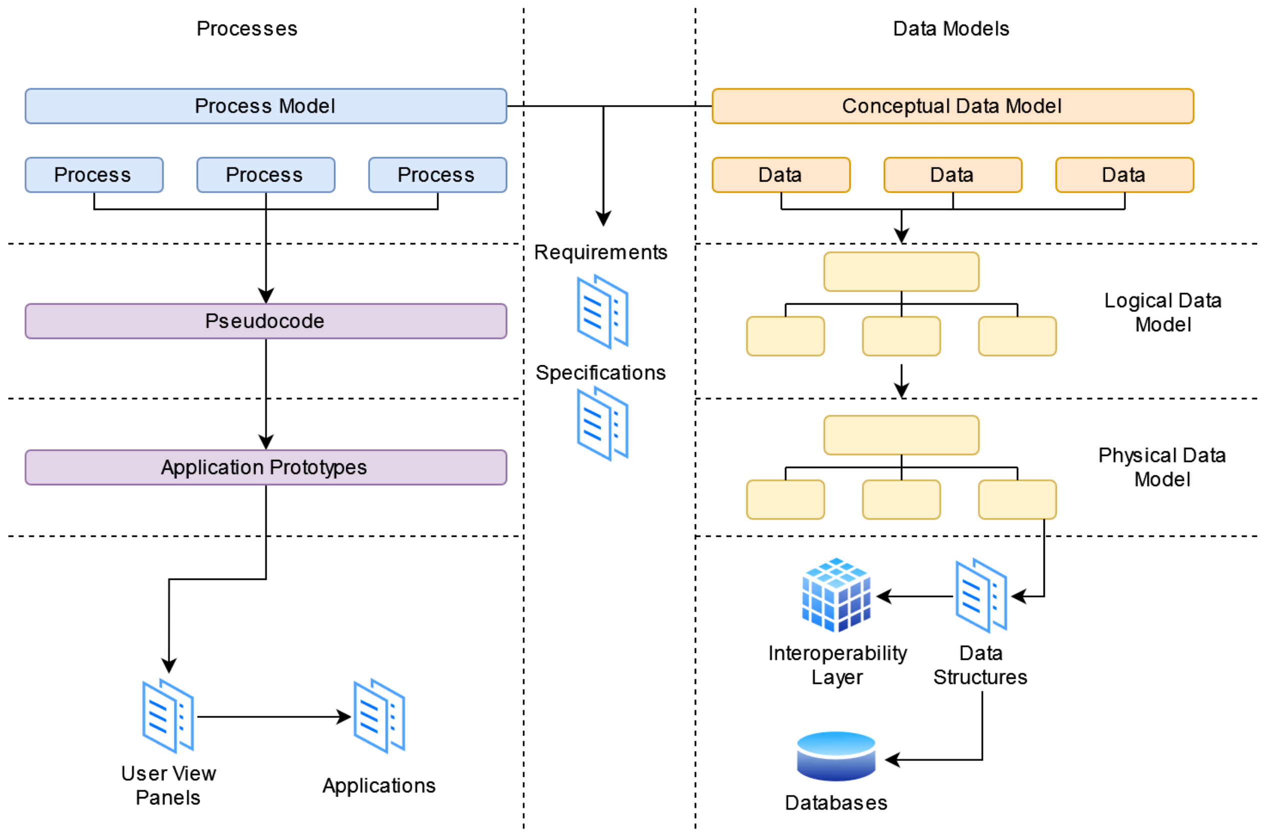


Figure 9: High-level presentation of the data modeling process.

The data modeling process will produce three different types of data model schemas – conceptual, logical, and physical - while progressing from the initially identified data objects and identified requirements to the actual implementation of data models inside the BRIGHT interoperability layer.

The identified data objects and requirements will be initially represented in the form of a conceptual data model. This will represent the initial technology independent specifications about the data and will be used as a tool to discuss all related requirements with the involved BRIGHT entities.

The conceptual data model will be translated into the logical data model in the following step. The logical data model will hold the specifications of data structures that can be implemented in the BRIGHT semantic adaptation component inside the BRIGHT interoperability layer and inside the involved databases.

In the final step in a data modeling process, physical data models will be created based on previously defined logical data models. Physical data models will organize data into tables, data access accounts, access specifications, data translation specifications etc.

The presented data modeling process and resulting data models are progressive. Data models will evolve during the following discussions and implementations to cover all needs of the BRIGHT interoperability layer and thus the BRIGHT system. The specifications of data models will change in response to changes and extensions of currently identified data objects and processes in a BRIGHT project.

3 Data Exchange Systems and Ontologies

3.1 SAREF

3.1.1 Overview

IoT fragmentation is one of the main threats in large-scale IoT adoption. To address this, the current fragmented landscape of IoT technologies requires standardised interfaces and data models to ensure interoperability. In this context, one of the main challenges to ensure interoperability is to have a set of standard data models that allow not only information but also the meaning of that information to be exchanged to avoid any misinterpretation between senders and receivers.
The SAREF (Smart Appliances REFerence) ontology [16] has been developed and standardized by the European Commission in close cooperation with ETSI (European Telecommunications Standards Institute) to provide a modular and domain-independent semantic layer for smart appliances. The core concepts of SAREF ontology are depicted in Figure 10.

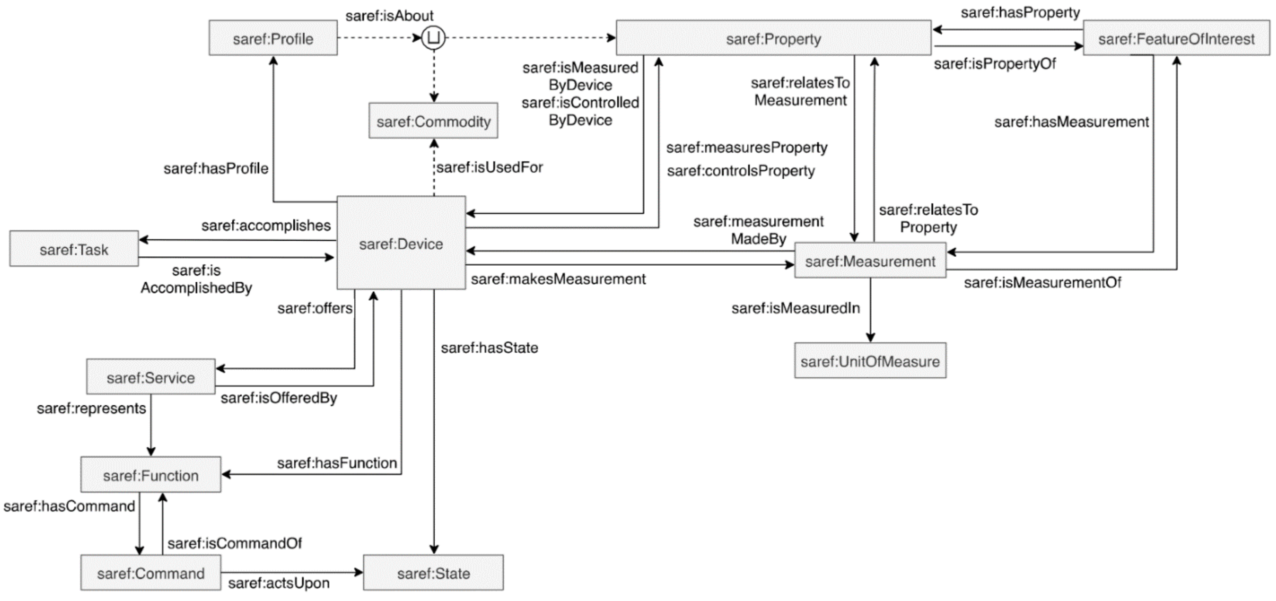


Figure 10. An overview of the main classes of SAREF and their relationships (Source: ETSI)

The starting point of the SAREF ontology is the concept of *Device* representing tangible objects designed to accomplish one or more *Tasks* in different types of locations and associated with *States*. The ontology offers a list of basic *Functions* that can be combined towards more complex functions in a single device. A *Service* can represent one or more functions offered by a device that wants its functions to be discoverable, registerable, and remotely controllable by other devices in the network. A device can be characterized by a *Profile* that can be used to collect information about a certain *Property* or *Commodity* (e.g. energy or water) for optimizing its usage in the home/building in which the device is located. Together with the *Measurement*, *Property* and *UnitOfMeasure*, the ontology allows to relate different measurements of a given device to different properties measured in different units.

3.1.2 Relevance to BRIGHT

In the BRIGHT project, SAREF ontology will be exploited for the development of the Interoperable home automation gateway tool as part of task 6.6. The home-IoT Gateway of DomX provides for monitoring and control of the home environment, integrating various home sensors/controllers/appliances belonging to different vendor ecosystems and making them interoperable.

DomX Interoperable gateway device will be extended to support SAREF ontology. To this purpose the SAREF4ENER [16] variant will be exploited. SAREF4ENER is an extension of SAREF for the Energy domain that was created in collaboration with Energy@Home [17] and EEBus [18], the major Italy and Germany-based industry associations, to enable the interconnection of their different data models. SAREF4ENER is an OWL-DL ontology that extends SAREF with 63 classes, 17 object properties and 40 data type properties. SAREF4ENER focuses on demand response scenarios, in which customers can offer flexibility to the Smart Grid to manage their smart home devices by means of a Customer Energy Manager (CEM). The CEM is a logical function for optimizing energy consumption and/or production that can reside either in the home gateway or in the cloud. Moreover, the Smart Grid can influence the quantity or patterns of use of the energy consumed by customers when energy-supply systems are constrained, e.g. during peak hours. SAREF4ENER is published as an ETSI tecnical specificatoion (ETSI TS 103 410-1).

3.2 S2 Communication (CEN-CENELEC EN50491-12 standard series)

3.2.1 Overview

The EN 50491-12-1 architecture focuses on the premises side of the smart grid and is mainly concerned with the communication between smart devices and CEM. Figure 11 provides a logical view of the components that can be found at the premises side.

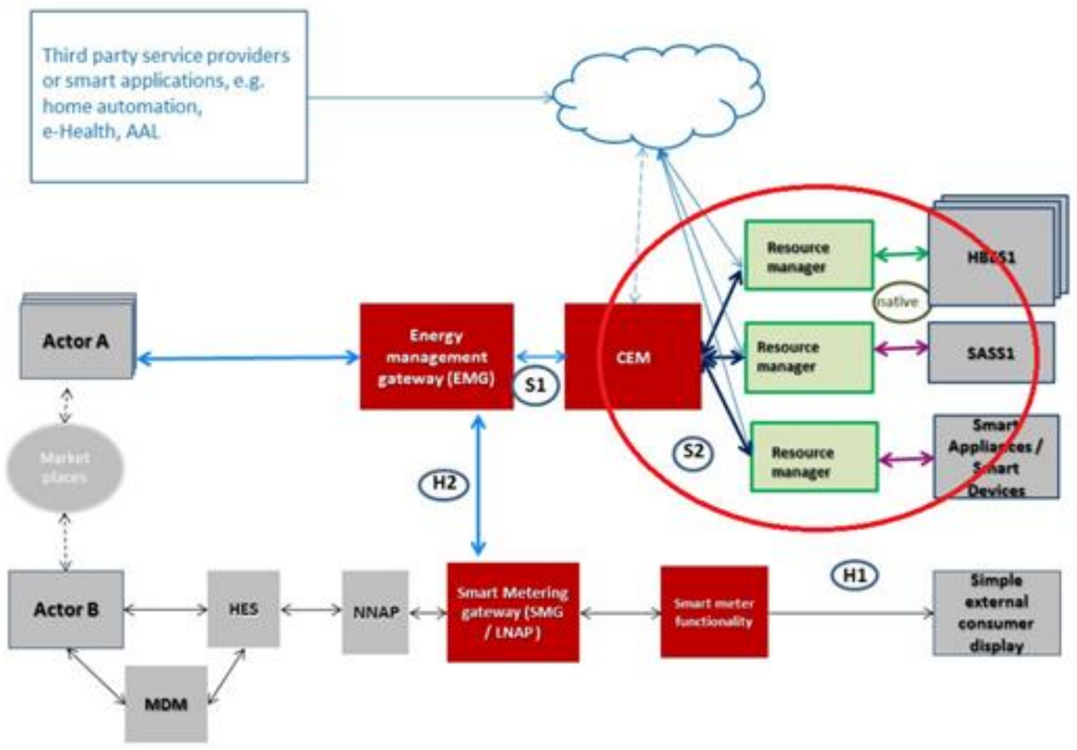

Third party service providers or smart applications, e.g. home automation, e-Health, AAL
Resource manager
HBES1
native
Actor A
Energy management gateway (EMG)
S1
CEM
Resource manager
SASS1
S2
Resource manager
Smart Appliances / Smart Devices
Market places
H2
H1
Actor B
HES
NNAP
Smart Metering gateway (SMG / LNAP)
Smart meter functionality
Simple external consumer display
MDM

Figure 11. The logical view of components at the premises with the S2 interface encircled.

The logical view shows all of the relevant smart grid systems on the premises, the red circle outlines the scope of CEN-CENELEC's 50491-12 standard series. Within this standard series, the so called S2 interface is being specified in the EN50491-12-2 standard [19].

The S2 interface is used to communicate the energy flexibility of smart devices to the Customer Energy Manager (CEM). The CEM also uses S2 to send instructions to smart devices to exploit their flexibility in a specific way. The components involved in the S2 communication are described below.

- **Smart Devices.** Smart Devices can offer energy flexibility by deviating from their normal consumption/production pattern. These devices can be controlled externally so that they can be integrated into the premises smart grid system. These devices are very diverse and perform a wide range of functions within a home or a building, such as white goods, PV, HVAC, etc. In Figure 11, this is reflected by the different terminology that is being used. Smart devices/appliances represent devices like white goods. The Home and Building Electronic System (HBES) are systems that are used in a home or building automation and perform functions such as switching, open and closed loop control. Single Application Smart System (SASS) are systems that are composed of a group of devices that work together for a single application. Think of an HVAC system that is composed of components such as fans, chillers, radiators etc. Controlling a single component within such a system for flexibility purposes might disrupt the correct functioning of the complete system. Therefore the entire system with all of its components should be treated as a single source of flexibility.

  As is apparent, these devices are very diverse in their functionality. This also goes for the protocols that are used to control these devices externally. Examples of such (IoT) protocols are KNX, EEBUS/SPINE, ModBus, Zigbee, Bluetooth, WiFi, Z-Wave, but also proprietary protocols. The same holds for the data models/parameters that are used. It is virtually impossible for a Customer Energy Manager to be aware of and support all possible permutations of functionality, protocols and data models. This is where the Resource Manager and the S2 interface come in.

- **Resource Manager.** The Resource Manager is an intermediary logical component that on one side communicates with the smart devices using its native protocol and data model and understands the functionality that the device performs. On the other side, it communicates the flexibility options of the devices to the Customer Energy Manager (CEM). The CEM is only interested in the flexibility that the device has to offer, not in all of the available detailed device parameters and protocols. These would simply overwhelm the CEM and would require adaptations to be made to the CEM every time a new device would be connected.

  The Resource Manager translates the low level device information into more high level information on the energy flexibility that is offered to the CEM via the S2 interface. This is not a straightforward mapping; information that is not relevant for energy flexibility needs to be filtered out while other information needs to be enriched to make it relevant for energy flexibility. Take a thermal buffer for example; a Resource Manager will have to understand what the capacity of that buffer is and how fast it can be heated. The S2 Control Types sections below describe in more detail which energy flexibility information is conveyed over the S2 interface. The Resource Manager will also receive instructions over S2 from the CEM to use the flexibility in a particular way.

In providing flexibility to the CEM, the Resource Manager will also take user comfort as well as the operational boundaries/safety margins of the device into account. These aspects will also be checked if the Resource Manager receives instruction from the CEM. If user comfort or the operational boundaries/safety margins are compromised by executing a CEM instruction it is the responsibility of the Resource Manager to reject that instruction.

- **Customer Energy Manager.** The CEM takes into account the flexibility that is being provided by all Resource Managers on the premises. Based on its optimization objectives and additional external information/incentives, it will decide how to use that flexibility to meet its objectives as closely as possible. Examples of CEM objectives could be to optimize dynamic energy tariffs, promote self-consumption as much as possible or to help the DSO alleviate congestion. After the CEM decides on how to use the flexibility, it will send instructions to the Resource Managers over S2.

  By using S2, a lot of the implementation details of the devices are hidden for the CEM and it can focus on its core business: managing energy flexibility. This enables the CEM to connect to a wide variety of devices with little effort, thus promoting interoperability.

S2 Control Types
Resource Managers are all capable (if supported by the underlying smart device) to provide power/energy measurements and forecasts. In addition to these basic and generic functions, the S2 interface features five control types representing different energy flexibility types. A Resource Manager will map the flexibility of the device it represents onto one of these control types. The CEM will only have to implement these control types to be able to connect to all devices via their respective Resource Managers. The control types are described below:

- Power Envelope Based Control. This control type is used for devices that can not be controlled by the CEM to adhere to a specific value for their production or consumption. They can, however, be asked by the CEM not to exceed certain power limits over time. A typical example of such a device would be a PV panel. The CEM cannot directly control its production as this is dependent on the amount of sunshine, but it can ask the PV panel not to exceed a certain production limit, also known as curtailment. This feature is very useful for congestion management for example. When there is too much production for the local grid to handle, this control type can be used to limit the output of the PV panel to a manageable level.

- Power Profile Based Control. The power profile based control type is typical for devices that perform a function with a corresponding power profile that is known or can be predicted beforehand. Their main flexibility comes from the ability to change the start time of that power profile. White goods, such as a washing machine with a delayed start option, are good examples of this category. For example, a consumer fills the washing machine with dirty clothes, selects a program and chooses the final time by which this program should be finished. The CEM can then decide what the best possible start time is, giving its optimization objectives.

  Another type of flexibility is offered by this control type is the ability to choose between multiple alternative power profiles. The heating cycle of the washing machine might have alternative profiles, e.g. one that consumes less power but requires more time to heat the water and one that consumes more power and takes less time to reach the target temperature. The CEM can then choose which one of these alternatives to use.

- Operation Mode Based Control. Devices that fall within this control type have the possibility to control the amount of power they produce or consume, without significant effects on their future flexibility options. Typical examples for this control type are diesel generators and variable electrical resistors. Such devices are often useful for balancing microgrids. Operation mode devices offer a lot of flexibility; they can assume a range of power levels at almost arbitrary moments in time. When this type of flexibility would be modelled with power profiles, as used for power profile based control, the number of possible permutations would rapidly grow beyond practical limits.

  To avoid such issues, the operation mode control type is modelled as a state machine. A resource manager can declare multiple operation modes for a device. An operation mode is a mode/state that a device can find itself in, that is associated with a specific power value. For example, a diesel generator can have three operation modes: one for being off, one for running at reduced power and one for running at full power. The “off” operation mode has a power value of 0 W associated with it, the “reduced power” operation mode has a power value of -1800 W (a negative value denotes production), and the “full power” operation mode has a power value of -3000 W.

  Transitions between operation modes are also explicitly specified. This way, the possible transitions between operation modes may be restricted. Transitions can also be equipped with timing constraints: a device can for example express that it needs to run for a minute in “reduced power”, before it can move on to “full power”. This can be achieved by defining a “minimum on time” timer that blocks the transition when its value is not equal to 0.

  The CEM can send instructions that will tell the Resource Manager which operation mode to go to next. These instructions also contain timestamps to inform the Resource Manager on when the transition to the next operation mode should be made.

- Fill Rate Based Control. The fill rate-based control type can be used for devices that have the ability to store or buffer energy. How energy is stored or buffered does not matter, as long as there is a means to measure how full the storage or buffer is.

  There are many examples of devices that can store or buffer energy. Stationary batteries and electric vehicles are examples of devices that store energy in batteries. Heating devices such as CHPs, (hybrid) heat pumps or boilers can buffer energy in a dedicated heat buffer (typically a thermally insulated water tank), but a room with an allowable bandwidth for the temperature can also be used as a buffer.

  Finally, there are also devices that produce cold, like air conditioners, fridges and freezers. Just like heat, cold can be buffered. There are even more ways to buffer or store energy imaginable, such as storing energy in the form of hydrogen, air pressure, water pressure or angular momentum.

The main component of this control type is the storage itself. A device shall be able to inform the CEM about its fill level, a measure of how full the storage is, and the lower and upper bounds that the fill level should remain within. If applicable, it can also inform the CEM about its target fill level and by when that should be reached. This would be useful when charging an EV for instance. In addition to the storage there are also actuators that can affect the fill level of the storage. E.g. an electrical heating element in a hot water buffer.

The behaviour of the actuators is described with a state machine, just like the operation mode based control type. In this case however the states also specify what their influence on the fill level of the buffer is.

- Demand Driven Based Control. Demand Driven Based Control can be used for systems that are flexible in the type of energy carrier they use, but are not capable of buffering or storing energy (in that case, Fill Rate Based Control should be used). A typical example is a hybrid heat pump, that generates heat using either electricity (using a heat pump) or natural gas (using a gas boiler), but doesn't have a thermal buffer. The hybrid heat pump must deliver a given amount of heat (hence demand driven), but can still decide whether to generate this heat using electricity or natural gas. Typically, such systems favour the heat pump, but use the gas boiler in case the heat demand cannot be fulfilled by the heat pump alone or when there is a shortage of capacity in the electricity grid.

  Similar to the Fill Rate Based Control, Demand Driven Based Control has the concept of multiple actuators. Again the behaviour of these actuators is described using a state machine. This time the states do not specify their influence of the fill level of the buffer is, but they specify a supply rate that can be matched with the demand. The CEM can select a state for each actuator as long as the demand is being matched by their aggregated supply.

3.3 DLMS/COSEM

3.3.1 Overview

In March 2009 European Commission issued Mandate M/441 which enabled standardization and interoperability of smart metering functionalities and communication in multi-utility systems and applications across Europe and thus improve customers' awareness of actual consumption and adaptation to their demands.

DLMS/COSEM is a global standard for smart meters to provide interoperability between different energy metering devices in an efficient and secure manner. The main components of the standard specify object-oriented data model, application layer protocol and communication profiles. DLMS/COSEM specification is developed and maintained by the DLMS User Association [20]. The result of its technical work is published in the public domain as DLMS UA Books, internationally is standardized under IEC 62056 DLMS/COSEM set of standards. Core standards are:

- IEC 62056-5-3, DLMS/COSEM application layer [21],
- IEC 62056-6-2, COSEM interface classes [22],
- IEC 62056-6-1, OBIS Object identification system [23].

Even though specifications have a common heading, "Electricity metering data exchange – The DLMS/COSEM suite" it is not specific only to electricity metering but also used for gas, water and heat metering.

If DLMS/COSEM is a language, then the semantics of the language is described by the COSEM object model and the syntax of the language is specified by DLMS.
Metering infrastructure in pilot 2 (see also Section 8.2) will use DLMS/COSEM compliant smart meter provided by Iskraemeco.

Even though smart meters are part of DSO's data collection systems its application layer, object models and communication interfaces are implemented in a way the third parties can access data that are relevant to BRIGHT. Data exchange between the metering device and collection system is based on the client-server relationship. The metering device acts as a server and the collection system acts as a client.

Physical smart meter device is called a physical device and consists of several logical devices. It is mandatory that each physical device contains management logical devices. The COSEM logical device contains a set of COSEM objects. Each logical device has its own address provided by the addressing scheme of the lower layers of the protocol stack. The logic device has its own name called COSEM logical device name (LDN) and the manufacturer ensures that LDN is unique.

Term object in COSEM is defined as a collection of attributes and methods. The first attribute of the object is the *logical_name* and is one part of the identification of the object. Object's logical name is described with OBIS codes. Another common attribute objects have is *value*.

An object can have a number of methods to, for example, set or reset the value of the attribute.

Common characteristics of objects are generalized as interface class and each such class has its own ID called *class_id*. Instantiations of such classes are called COSEM interface objects.

Example of interface class and its instances is shown in Figure 12.

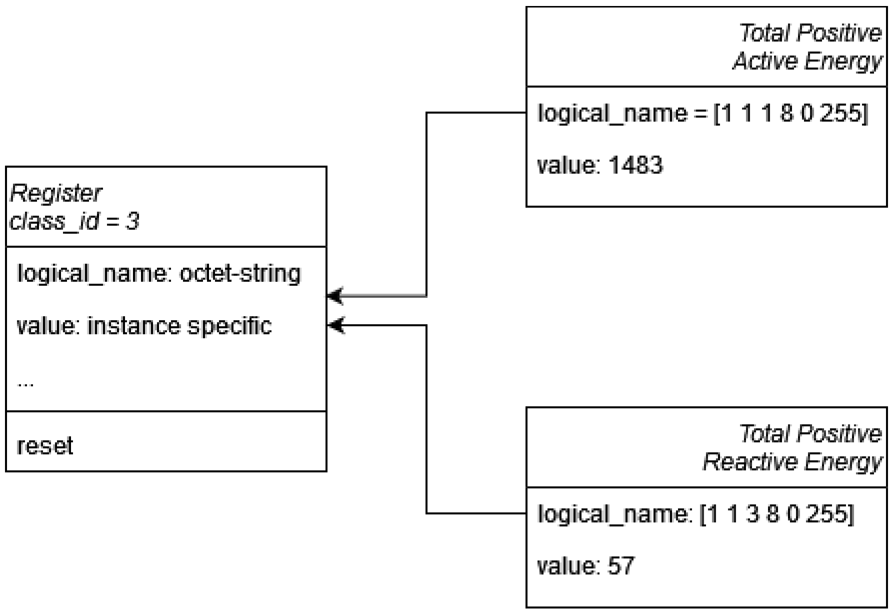

Total Positive Active Energy
logical_name = [1 1 1 8 0 255]
value: 1483
Register class_id = 3
logical_name: octet-string
value: instance specific
...
reset
Total Positive Reactive Energy
logical_name: [1 1 3 8 0 255]
value: 57

Figure 12. Example of interface class and it's instances.

Where interface class presented is Register with two objects Total Positive Active Energy and Total Positive Reactive Energy. Interface class overview is presented with a table with the included class name, the attributes and the methods, Table 5. Each attribute and method is described in detail.

Table 5. Interface class with attributes and methods

Class name		**Cardinality**	**class_id, version**			
Attributes		**Data type**	**Min**	**Max**	**Def**	**Short name**
1. logical name	(static)	octet-string				x
2. …	(…)	…				x + 0x…
Specific methods (if required)		**m/o**				
1. …	(…)	…				x+ 0x…
2. …	(…)	…				x + 0x…

In order for a client to access COSEM objects in the server, an application association must be established. This will identify both partners and characterize the context within which associated applications will communicate. Application associations are modelled by special COSEM objects. For example, the server may grant different access rights to clients, some COSEM objects can or can not be seen or have access to attributes and methods. The client can obtain the list of visible COSEM objects by reading the *object_list* attribute of the appropriate association object.

BRIGHT will not have access to DSO's collection system nor it will replace it and for this reason, further detailed description of DLMS/COSEM is out of the scope of this document.
All additional descriptions of meter data and data presentation relevant to BRIGHT in the meter as well interface to data will be described in chapter CIP.

4 APIs

Various APIs are needed to exchange the data between different services or other components in the BRIGHT project. The APIs should cover not only high-level connection among services, databases, visualization dashboards, query engines, etc. They should also cover APIs covering communication on “low-level” transmitting raw data such as communication with SM. To guarantee full interoperability, all the developed tools should use common APIs defined in this section.

4.1 REST API

4.1.1 Introduction

REST is a set of architectural constraints that was created to guide the design and development of the architecture for the World Wide Web. REST defines a set of constraints for how the architecture of an Internet-scale distributed system should behave. The REST architectural fashion emphasises the scalability of interactions between components, uniform interfaces, independent deployment of components, and the creation of a layered architecture to facilitate caching components to reduce user-perceived latency, enforce security, and encapsulate legacy systems. REST has been employed throughout the software industry and is a widely accepted set of instructions for creating stateless, reliable web APIs. A web API that obeys the REST constraints is informally described as RESTful. RESTful web APIs are typically loosely based on HTTP methods to access resources via URL-encoded parameters and the use of JSON or XML to transmit data. When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP: JSON, HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it’s language-agnostic, as well as readable by both humans and machines.

4.1.2 Relevance to BRIGHT

In BRIGHT project most involved resources are actually being accessed through the REST API, since it is the most common architecture employed by the majority of vendors/developers/systems. Specifically, domX is using the REST API to define the format of the resources, as can be seen in the following subsection in detail.

4.1.3 Implementation aspect

A brief example of the API developed and used by domX in order to interact with the smart Heating Controllers that are deployed in the homes of individual users is described below.

In the first example, we demonstrate the GET command to acquire info regarding the available devices. By issuing the command:

- `GET:` http://<domain>/api/v1/devices

We receive the response:

```
[
   {
     "deviceid": "domx_ot_f4:ff:ff:ff:ff:ff",
       "userid": 1,
       "registeredat": "2020-11-03T14:38:56.075Z",
       "latitude": "52.663158",
       "longitude": "32.9477498",
       "tags": null,
        "device_type": 1,
        "device_name": "test home",
        "region": "Thessaloniki, Greece",
        "updatedat": "2020-11-03T14:38:56.075Z",
        "boiler_make": "ARISTON",
        "boiler_model": "CLAS ONE 30"
    }

```

Apparently, a specified format is used to consolidate the requested data, including both the parameters names as well as the requested values, thus the data can be visible both by a machine and human.

In the second example we issue a GET command to retrieve data by a specific device:

- `GET:` http://<domain>/api/v1/devices/<deviceId>/data

While, the response of the above request is:

```
{
    "measurements": {
        "heat": {
            "name": "heat",
            "data_type": "boolean"
        },
        "water": {
            "name": "water",
            "data_type": "boolean"
        },
        "fault": {
            "name": "fault",
            "data_type": "boolean"
        }
    },
    "actions": {
        "heat_set": {
            "data_type": "boolean",
            "action_type": "ot"
        },
        "water_set": {
            "data_type": "boolean",
            "action_type": "ot"
        },
        "t_set": {
            "data_type": "float",
            "action_type": "ot",
            "min": 0,
            "max": 100
        },
        "t_r_set": {
            "data_type": "float",
            "action_type": "ot",
            "min": 16,
            "max": 32
        },
        "t_dhw_set": {
            "data_type": "float",
            "action_type": "ot",
            "min": 0,
            "max": 100
        },
        "otc_cur": {
            "data_type": "float",
            "action_type": "ot",
            "min": 0,
            "max": 10
        },
        "force_update": {
            "data_type": "string",
            "action_type": "device",
            "values": [
                "dev",
                "staging",
                "live"
            ]
        },
        "devicereset": {
            "data_type": "boolean",
            "action_type": "device"
        },
        "factoryreset": {
            "data_type": "boolean",
            "action_type": "device"
        },
        "bypass": {
            "data_type": "integer",
            "action_type": "ot",
            "min": 0,
            "max": 2
        }
    }
}
```

The retrieved data are separated into two main categories "`measurements`" and "`actions`", with each parameter being defined in terms of type and possible values/range.

4.2 Streaming API

4.2.1 Introduction

Data streaming, also known as Event Streaming or Stream Data Processing, is a term that identifies the ability of a system to intercept multiple streams of data and perform a series of operations on them. Compared to the 'traditional' way of managing data, this approach avoids asynchronous processing (batch processing), thus avoiding the need to load data into special structures for subsequent analysis. One of the most popular tools for working with streaming data is Apache Kafka. Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time. Kafka provides three main functions to its users:

1. To **publish** (write) and **subscribe to** (read) streams of events, including continuous import/export of your data from other systems.
2. To **store** streams of events durably and reliably for as long as you want.
3. To **process** streams of events as they occur or retrospectively.

And all this functionality is provided in a distributed, highly scalable, elastic, fault-tolerant, and secure manner.

4.2.2 Apache Kafka core concepts

The first thing that everyone who works with streaming applications should understand is the concept of the **event**. An **event** records the fact that "something happened" in the world or in your business. It is also called a record or message in the documentation. When you read or write data to Kafka, you do this in the form of events. Conceptually, an event has a key, value, timestamp, and optional metadata headers.

Events are constantly written by **producers**. **Producers** are those client applications that publish (write) events to Kafka. **Consumers** are entities that use data (events). In other words, they can receive data written by producers and use this data. In Kafka, producers and consumers are fully decoupled and agnostic of each other, which is a key design element to achieve the high scalability that Kafka is known for. For example, producers never need to wait for consumers. Kafka provides various guarantees such as the ability to process events exactly-once. Kafka is the middleman between applications that generate data and applications that consume data.

Events are organized and permanently stored in **topics**. A topic can be compared to a folder in a filesystem, and the events are the files in that folder. Topics in Kafka are always multi-producer and multi-subscriber: a topic can have zero, one, or many producers that write events to it, as well as zero, one, or many consumers that subscribe to these events. Events in a topic can be read as often as needed—unlike traditional messaging systems, events are not deleted after consumption. Instead, you define for how long Kafka should retain your events through a per-topic configuration setting, after which old events will be discarded.

Topics are **partitioned**, meaning a topic is spread over a number of "buckets" located on different Kafka brokers. This distributed placement of your data is very important for scalability because it allows client applications to both read and write the data from/to many brokers at the same time.
Figure 13 shows an example of a topic. In the example, the topic has four partitions, P1–P4. Two different producer clients are publishing, independently from each other, new events to the topic by writing events over the network to the topic's partitions. Events with the same key (denoted by their color in the figure) are written to the same partition. Note that both producers can write to the same partition if appropriate.

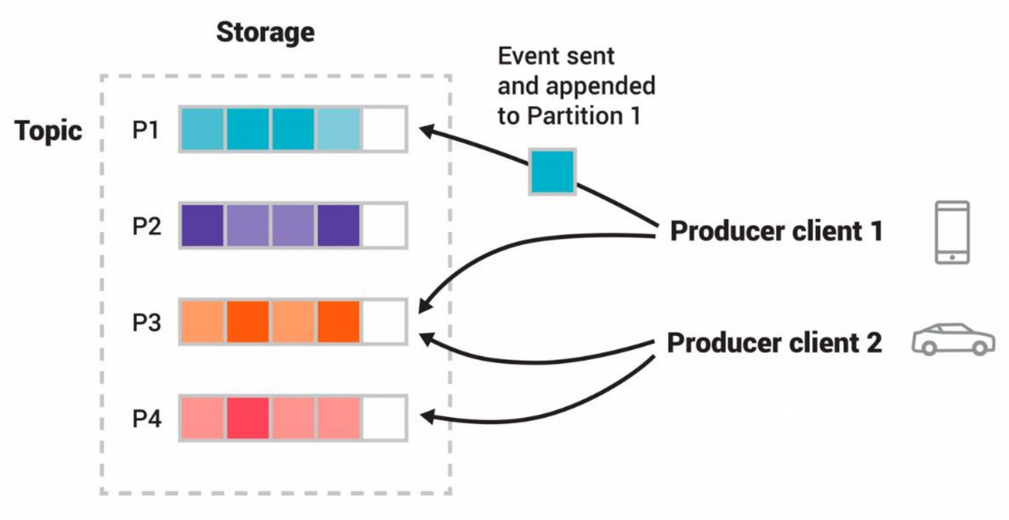


Figure 13. Topics and partitions in Kafka

To make data fault-tolerant and highly-available, every topic can be **replicated**, even across geo-regions or datacenters, so that there are always multiple brokers that have a copy of the data just in case things go wrong, you want to do maintenance on the brokers, and so on. A common production setting is a replication factor of 3, i.e., there will always be three copies of your data. This replication is performed at the level of topic-partitions.

4.2.3 Relevance to BRIGHT

In BRIGHT Kafka will be used for the implementation of the Message queue system as part of the interoperability layer. BRIGHT's architecture follows a microservices approach. One of the first big considerations you'll make when building up a microservice architecture is whether to have the services communicate directly with one another or to use a broker system. Adopting the broker model makes the system more flexible and resistant to failure, this is why we are using Kafka inside BRIGHT. The goal of Apache Kafka is to solve the scaling and reliability issues that hold older messaging queues back. A Kafka-centric microservice architecture uses an application setup where BRIGHT's tools communicate with each other using Kafka as an intermediary. This is achievable thanks to Kafka's publish-subscribe approach for handling record writing and reading. The publish-subscribe model (pub-sub) is a communication strategy in which the sender sends events — whenever events are available — and each receiver chooses which events to receive asynchronously.

5 Protocols

5.1 MQTT

5.1.1 Introduction

MQTT is a publish-subscribe network protocol suitable for the Internet of Things (IoT). It is designed to be lightweight with a small code footprint and minimal network bandwidth requirements. It is ideal for connecting small remote devices on unreliable and low-bandwidth connections to edge devices or datacenters. The protocol is usually implemented on top of a TCP/IP network stack, but can run on any network protocol that provides ordered, lossless and bi-directional communication.

5.1.2 MQTT core concepts

Using MQTT protocol two types of network entities are defined in a MQTT protocol standard:

- MQTT broker
- MQTT client

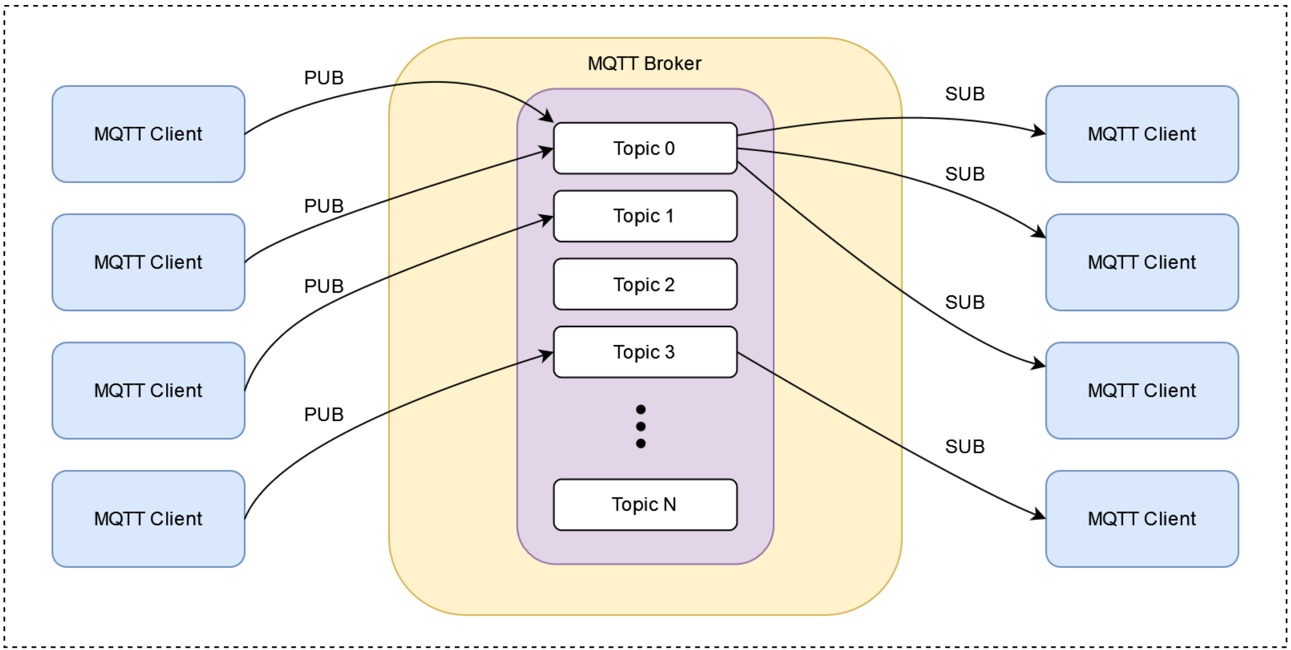


Figure 14. Example of MQTT network.

MQTT broker is a server that receives all messages from the MQTT clients (publishers) and forwards them to the subscribed MQTT clients (subscribers). Information that flows through the MQTT broker is hierarchically organized into topics. Each MQTT client that connects to the MQTT broker can subscribe to one or more MQTT topics. When new data is published on a particular topic, the MQTT broker distributes the data to all subscribed MQTT clients. If there is no subscribed MQTT client for a particular topic, the MQTT broker discards the published data.

MQTT client is any device that runs MQTT protocol and can connect to a MQTT broker. MQTT client code footprint is very small and can be implemented on small embedded devices with limited processing and memory resources. Many implementations of MQTT client protocol can be found (also many device-specific implementations) that are implemented for different platforms and in many programming languages. A device-specific implementation can be found for Arduino devices, ESP32 devices, mbed devices and many more [24]. Some of the popular implementations that run on many POSIX and Windows devices are for example Eclipse Paho C [25] and libmosquitto [26].

MQTT protocol also specifies quality of service (QoS) measure, where for each connection between a MQTT client and a MQTT broker a level of QoS can be specified. Three QoS settings are specified and they differ in the amount of communication overhead they impose according to the setting:

- Level 0: at most once – the message is sent only once without the delivery acknowledge (fire and forget).

- Level 1: at least once – the message is being retransmitted multiple times until receive acknowledge is received (acknowledged delivery).

- Level 2: Exactly once – the sender and receiver use a two-level handshake to ensure only one copy of the message is received (assured delivery).

5.1.3 MQTT security

Most of MQTT protocol implementations use plain TCP/IP transport, where all communication content can be accessed by any participant alongside the chain of devices between the sender and the receiver. Any agent alongside the communication chain can read the content of the TCP packet and can even modify it.

To provide a secure communication channel between the MQTT client and MQTT broker, MQTT protocol uses Transport Layer Security (TLS). TLS ensures the content of sent messages can not be read or altered by any third party device or agent.

MQTT protocol uses username and password as a client authentication mechanism to establish a secure encrypted connection. MQTT broker provides a server X.509 certificate, which should be issued by a trusted authority, and is used by the client to verify the identity of the server. In addition to username and password, client identification can be extended by using unique device identifiers (e.g. Universal Unique Identifier - UUID) or client X.509 certificates. The client presents the client certificate to the broker during the TLS handshake.

5.1.4 Relevance to BRIGHT

Inside the BRIGHT pilots, MQTT will be the main communication interface for many deployed devices for collecting measurements and also for controlling the connected end devices. Many of those devices are connected to the cloud and management systems using low bandwidth IoT connections and mobile internet connections, where low data usage is of great importance. MQTT will also provide a secure and standardized communication interface for all connected devices using MQTT to the BRIGHT interoperability layer.

5.2 CIP (I1)

5.2.1 Introduction

I1 interface of a smart meter is used as Consumer Information Push (CIP) local port. It is a unidirectional port (from meter to consumer) and provides relevant metering data to the consumer such as (but not limited to): active, reactive, apparent power and energy used (current and average), voltages, currents, load profiles, demand power, water-, gas-, heating- energy registers, etc.

CIP protocol stack can be either HDLC based or IP based protocol stack with physical characteristics defined in companion standard used in Dutch Smart Meter Requirements [27].

Smart meters in Pilot 2 will use an HDLC-based protocol stack with a predefined set of data to transmit to the consumer. For the purposes of GDPR, CIP data can be encrypted. This set of data will be defined at a later stage of the project.

The physical connector is type RJ12; the metering device holds a female connector. Pin assignment of P1 interface is shown in the following table

Table 6. Pin assignment on P1 connector

Pin #	Signal	Description	Note
1	+5V	Power supply	Not available in older versions (max 250 mA)
2	DR	Data request	Input (active high)
3	DGND	Data ground	
4	NC	Not connected	
5	Data	Data line	Output (open collector)
6	PGND	Power ground	

All signals are galvanically isolated from the mains.

Figure 15 shows the RJ12 male plug and position of the first pin, and Figure 16 shows the functional block diagram of the P1 connector.

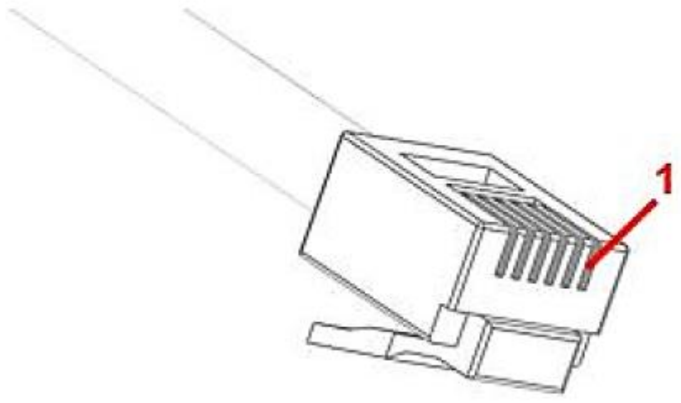


Figure 15. P1 connector on RJ12 male plug.

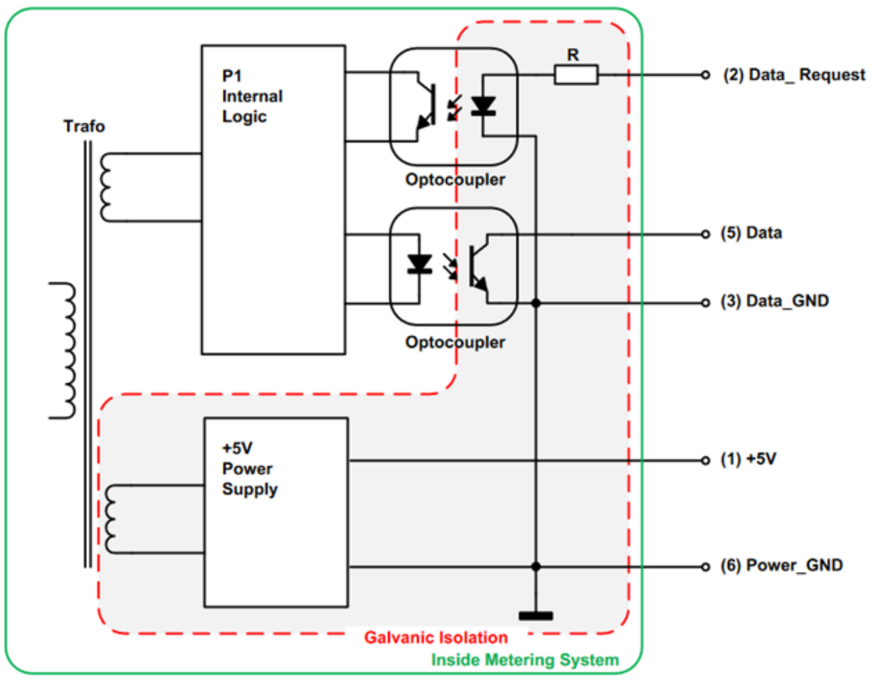


Figure 16. Functional block diagram of P1 connector.

Smart meters used in BRIGHT will use CIP protocol stack set to HDLC, frame type 3 and the non-basic frame format transparency according to IEC 13239, sec- 4.3.3. Encoding is ASN.1

Sensitive user information transmitted from the meter will be secured (encrypted and secured). Security material used in the process is independent of security material used in smart meter – DSO communication.

6 Conclusions

This deliverable introduces and defines the data interoperability layer and specifies requirements the BRIGHT service should follow. An overview of the most relevant interoperability concepts for the BRIGHT system is presented and discussed. The concept of data models and ontologies are introduced and placed in a broader sense of the BRIGHT project. A list of identified and harmonized data objects is presented that will be used and adapted to the data models and ontologies used by BRIGHT consortium. This will lead to the creation of BRIGHT specific ontology, created by using and extending existing ontologies. This will enable the semantic adaptation inside the BRIGHT interoperability layer and the service and data interoperability inside the BRIGHT project.

In the second part of the document, existing data exchange systems and ontologies (SAREF, S2, DLMS/COSEM) that are going to be used inside the BRIGHT project are presented and discussed, followed by the presentation of the APIs used inside the BRIGHT project (REST API, Apache Kafka streaming API). The Apache Kafka will act as a basis for scalable and secure communications inside the BRIGHT framework by means of providing high-performance data pipelines, data integration, semantic adaptation, and access to heterogeneous data sources. Finally, communication protocols for data collection are presented like publish-subscribe protocol MQTT and CIP protocol for collecting data from smart meters.

For the next steps, the harmonized inputs of the complete architecture as defined in D2.3 will be used to align the interoperability architecture with BRIGHT architecture. Moreover, from WP3 the data objects that need to be additionally modeled or modified will be identified. This will enable to incorporate the indirect input from citizens obtained with various social engagement strategies as defined in WP3. Finally, the D4.1 input gives us the first feedback on which technologies are feasible for incorporating them in the BRIGHT interoperability layer, thus creating the final architecture of the interoperability layer. This along with the BRIGHT specific ontology will be reported in the next version of this deliverable (D2.5) scheduled for M19.

The output of this deliverable and the one following is giving the requirements that need to be followed by technical WPs for addressing interoperability issues when creating new solutions.

References

[1] Laney D. 3D Data Management: Controlling Data Volume, Velocity, and Variety. META Group Research Note 2001.
[2] Niculescu V. On the Impact of High Performance Computing in Big Data Analytics for Medicine. Applied Medical Informatics 2020;42:9–18.
[3] Spyns P, Meersman R, Jarrar M. Data modelling versus ontology engineering. SIGMOD Rec 2002;31:12–7. https://doi.org/10.1145/637411.637413.
[4] Sung-Kook H. Towards Semantic oriented Database - ppt download n.d. https://slideplayer.com/slide/14432021/ (accessed September 10, 2021).
[5] soa 2021. https://www.ibm.com/cloud/learn/soa (accessed September 10, 2021).
[6] Mats'ela M. Microservices Architecture vs. Service Oriented Architecture. Mualle Mats'ela 2019. https://mualle.net/microservices-architecture-vs-service-oriented-architecture/ (accessed September 10, 2021).
[7] Microservices vs SOA: What's the Difference? Tiempo Development 2020. https://www.tiempodev.com/blog/microservices-vs-soa/ (accessed September 10, 2021).
[8] Enterprise service bus. Wikipedia 2021.
[9] Delgado J. Service Interoperability in the Internet of Things. In: Bessis N, Xhafa F, Varvarigou D, Hill R, Li M, editors. Internet of Things and Inter-cooperative Computational Technologies for Collective Intelligence, Berlin, Heidelberg: Springer; 2013, p. 51–87. https://doi.org/10.1007/978-3-642-34952-2_3.
[10] Lewis GA, Morris E, Simanta S, Wrage L. Why Standards Are Not Enough to Guarantee End-to-End Interoperability. Seventh International Conference on Composition-Based Software Systems (ICCBSS 2008), 2008, p. 164–73. https://doi.org/10.1109/ICCBSS.2008.25.
[11] Noura M, Atiquzzaman M, Gaedke M. Interoperability in Internet of Things: Taxonomies and Open Challenges. Mobile Netw Appl 2019;24:796–809. https://doi.org/10.1007/s11036-018-1089-9.
[12] Marcos E, Marcos A. A Philosophical Approach to the Concept of Data Model: Is a Data Model, in Fact, a Model? Information Systems Frontiers 2001;3:267–74. https://doi.org/10.1023/A:1011460711754.
[13] Taylor D. Data Modelling: Conceptual, Logical, Physical Data Model Types n.d. https://www.guru99.com/data-modelling-conceptual-logical.html (accessed September 13, 2021).
[14] Technologies M. Data Modeling Examples | What is Data Modeling 2021. Mindmajix 2021. https://mindmajix.com/data-modeling-examples (accessed September 13, 2021).
[15] What is Relational Data Model? Characteristics, Diagram, Constraints, Advanatges & Disadvantages. Binary Terms 2019. https://binaryterms.com/relational-data-model.html (accessed September 13, 2021).
[16] ETSI TS 103 264 n.d. https://www.etsi.org/deliver/etsi_ts/103200_103299/103264/02.01.01_60/ts_103264v020101p.pdf (accessed March 1, 2017).
[17] Energy@home - Home n.d. http://www.energy-home.it/SitePages/Home.aspx (accessed October 18, 2021).
[18] kennzeichen. EEBUS | Make the world speak energy. EEBus Initiative eV n.d. https://www.eebus.org/ (accessed September 23, 2020).
[19] DSF/PREN 50491-12 - General requirements for Home and Building Electronic Systems (HBES) and Building Automation and Control Systems (BACS) - Part 12: Smart grid - Application specification - Interface and framework for customer | Joinup n.d.

https://joinup.ec.europa.eu/collection/ict-standards-procurement/solution/dsfpren-50491-12-general-requirements-home-and-building-electronic-systems-hbes-and-building (accessed October 19, 2021).

[20] DLMS: Device Language Message Specification | dlms n.d. https://www.dlms.com/ (accessed September 30, 2021).

[21] IEC 62056-5-3:2017 | IEC Webstore | cyber security, smart city n.d. https://webstore.iec.ch/publication/27065 (accessed September 30, 2021).

[22] IEC 62056-6-2:2017 | IEC Webstore n.d. https://webstore.iec.ch/publication/34317 (accessed September 30, 2021).

[23] IEC 62056-6-1:2017 | IEC Webstore n.d. https://webstore.iec.ch/publication/32782 (accessed September 30, 2021).

[24] Software n.d. https://mqtt.org/software/ (accessed October 21, 2021).

[25] Craggs I. Eclipse Paho | The Eclipse Foundation n.d. https://www.eclipse.org/paho/index.php?page=clients/c/index.php (accessed October 21, 2021).

[26] Eclipse Mosquitto. Eclipse Mosquitto 2018. https://mosquitto.org/ (accessed October 21, 2021).

[27] P1 Companion Standard 2021. https://www.netbeheernederland.nl/_upload/Files/Slimme_meter_15_a727fce1f1.pdf (accessed September 10, 2021).

7 Annex 1

7.1 Data template Pilot 1

Location: Belgium
Owner: DuCoop, Participating partners: IMEC, CEN

Information object	Attributes	Granularity	Communication frequency (range)	Communication protocol	Data sharing format	Nature of data	Comments
Smart meter	Voltage (1 and 3 phase), Current (1 and 3 phase), consumption (2 tariffs), injection (2 tariffs), ID, Timestamp	3 min	5-10 sec	NEN-EN-IEC 62056-21	JSON	Sensitive	Electricity
Energy meter	Current, frequency, power, voltage, Consumed Energy (day, night and total), Timestamp	5 min	< 5 sec	MBUS, PROFINET, ModbusTCP	JSON	Sensitive	Electricity
Energy meter	Consumed energy, Timestamp	5 min	5 min	MBUS	JSON	Sensitive	Heat
Generation	Power, voltage (3 phases), current (3 phases), frequency, total yield, grid power, Timestamp	5 min	1 min	HTTPS	REST API being developed (JSON)	Open	PV, 5 installations, total of approx. 90 kWp
Storage	Timestamp, SoC, (dis)charging power, operational state	1 min	1 min	MQTT	REST API being developed (JSON)	Open	Battery storage, 240 kWh
EV charging	Total charging power (3	1 min	1 min	Modbus	REST API being	Open	

stations	phases), total current (3 phases), total voltage (3 phases), Timestamp				developed (JSON)		
Generation	Operational state, generated thermal power, Timestamp	1 min	1 min	Modbus	REST API being developed (JSON)	Open	Heat pump with three compression stages (20, 40, 60 kW)
Load	Consumed power, Timestamp	1 min	1 min	Modbus	REST API being developed (JSON)	Open	Heat pump with three compression stages (20, 40, 60 kW)
Weather	Date, Time, Temperature, Precipitation, Humidity, wind direction, wind speed, Solar radiation, UV index	1 min	1 min	ModbusTCP	REST API being developed (JSON)	Open	On-site weather station
Private living units	Temperature rooms, temperature set point rooms, district heating temperature, district heating return temperature, district heating flow, tapping temperature	5 sec	1 min	HTTP, REST API	REST API being developed (JSON)	Sensitive	

7.2 Data template Pilot 2

Location: Slovenia
Owner: SUN, Participating partners: ISKRA, COM

Information object	Atributes	Granularity	Communication frequency (range)	Communication protocol	Data sharing format	Nature of data	Comments
Smart meter	Voltage, Active power, Current, 1 & 3 phase, ID, Timestamp	15 min	On demand	HTTP	JSON	Sensitive	
Energy meter	Input Temperature, output Temperature, Energy, Active power Timestamp	1 s	On demand	HTTP	JSON	Sensitive	Heating and district heating
Generation	energy power, ID, Timestamp	15 min	On demand	HTTP	JSON	Sensitive	PV
Storage	temperature, Min. Temp., Max. Temp., Volume Timestamp, ID	1s, 15 min	On demand	HTTP	JSON	Sensitive	Heat storage

EV charging station	station ID, Voltage, Current, Real-time power, Price, Timestamp	tbd	tbd	tbd	tbd	Sensitive	Under construction not yet integrated in the system
Load	Voltage, Active power, Current, 3-phase, ID, Timestamp	15 min	On demand	HTTP	JSON	Sensitive	Complete building consumption
Weather	Date, Time, Temperature, Precipitation, Solar radiation	tbd	tbd	Tbd	Tbd	tbd	
Calendar	Working days, Weekend, Public holidays and bank holidays						
Property and measurements	ID, Timestamp, Unit, Value	15 min	On demand	HTTP	JSON	Sensitive	Rooms and corridors temperature, humidity, C02

7.3 Data template Pilot 3

Location: Italy
Owner: ASM, Participating partners: ENG, EMOT, COM

Information object	Attributes	Granularity	Communication frequency (range)	Communication protocol	Data sharing format	Nature of data	Comments
Smart meter	Voltage, Active power, Current, One phase, ID, Timestamp, Consumed Energy, Timestamp	5 seconds	5 seconds	MQTT	CSV	Proprietary	The smart meter are related to the customers/prosumers scattered in the city of Terni.
Generation	ID, Measurement unit, Timestamp, Generated active energy	5 seconds	5 seconds	MQTT	CSV	Proprietary	PV of 185 kWp.
Storage	Timestamp, SoC	5 seconds	5 seconds	MQTT	CSV	Proprietary	Type of storage, e.g. battery
EV	EV ID, Timestamp, SoC	2 seconds	2 seconds	MQTT	JSON (real-time data), CSV (historical data)	Proprietary	
EV charging station	Charging station ID, Socket ID,	1 second	1 second	MQTT	JSON (real-time data), CSV (historical data)	Proprietary	

	Nominal max power, Nominal min power, Real-time power, Timestamp						
BEMS (Building Energy Management System)	Zones temperature, humidity levels, technical setpoiints, timestamp	15 min	15 min	MQQT	CSV	Proprietary	The main loads of ASM district are managed by BEMS (HVAC, lighting system)
Weather	Date, Time, Temperature, Precipitation, Solar radiation	Daily	Monthly	Web site	CSV	Open	Umbria region web site
Calendar	Working days, Weekend, Public holidays and bank holidays						

7.4 Data template Pilot 4

Location: Greece
Owner: WVT, Participating partners: DOMX

Information object	Atributes	Granularity	Communication frequency (range)	Communication protocol	Data sharing format	Nature of data	Comments
Smart meter	Voltage, Active power, Current, One phase, ID, Timestamp	15 min	On demand/next day	JSON, CSV	XLSX, CSV, JSON	Sensitive/Proprietary	
Energy meter	Consumed Energy, Timestamp	15 min	On demand/next day	JSON, CSV	XLSX, CSV, JSON	Sensitive/Proprietary	
Load	Consumed electric energy, Timestamp	15 min	On demand/next day	JSON, CSV	XLSX, CSV, JSON	Sensitive/Proprietary	
Smart Heating Controller (domX)	Indoor/outdoor temperature, Space heating temperature setpoint, Domestic hot water temperature Setpoint, Boiler water temperature, DHW temperature, Heating usage, Hot water usage, Boiler modulation, Gas consumptio, Timestamp	1-5 min	On demand	HTTPS	JSON	Sensitive	Most attributes are provided every minute apart from the consumption data that are provided every 5’
Energy meter (domX)	Active Power, Voltage, Consumed Energy, Reactive Power, Power Factor, Timestamp	30 sec - 5 min	On demand	HTTPS	JSON	Sensitive	Most attributes are provided every 30’’ apart from

							the Energy Consumption that is provided every 5’