



Renewable Energy
for Self-Sustainable
Island Communities

WP6 - REACT platform integration and interoperability

D6.3 Semantic data model and connectivity with smart-grid



DISCLAIMER

The opinion stated in this report reflects the opinion of the authors and not the opinion of the European Commission.

All intellectual property rights are owned by REACT consortium members and are protected by the applicable laws. Reproduction is not authorised without prior written agreement.

The commercial use of any information contained in this document may require a license from the owner of that information.

ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 824395.

Project Data	
Project Acronym	REACT
Project Title	Renewable Energy for self-sustAinable island CommuniTies
Grant Agreement number	824395
Call identifier	H2020-LC-SC3-2018-2019-2020
Topic identifier	LC-SC3-ES-4-2018 Decarbonising energy systems of geographical Islands
Funding Scheme	Innovation action (IA)
Project duration	48 months (From 1 st of January of 2019)
Coordinator	VEOLIA – Ferran Abad
Website	http://react2020.eu
Deliverable Document Sheet	
Deliverable No.	D6.3
Deliverable title	Semantic data model and connectivity with smart-grid services
Description	Report and software describing the semantic data model and REACT connection with smart-grid services
WP No.	6
Related task	T.6.3 – Semantic data model and smart-grid connectivity
Lead Beneficiary	TEKNIKER (TEK)
Author(s)	Iker Esnaola-Gonzalez (TEK)
Contributor(s)	Ignacio Lázaro, Francisco Javier Diez (TEK), Dejan Paunovic, Dea Pujic, Lazar Berbakov, Nikola Tomasevic (IMP), Lluís Millet, Marco Mittelsdorf (FHG), James Freeman, Daniel Coakley (MERCE), Paulo Lissa (NUI Galway)
Type	Report, OTHER
Dissemination L.	Public
Language	English – GB
Due Date	30/06/2021
Submission Date	25/11/2021

VER	Action	Owner	Date
0.1	TOC definition	Iker Esnaola-Gonzalez - TEK	21/01/2021
0.2	TOC review	Nikola Tomasevic - IMP	18/02/2021
0.3	IMP initial contributions	Dejan Paunovic - IMP	06/05/2021
0.4	TEK initial contributions	Iker Esnaola-Gonzalez - TEK	21/06/2021
0.5	Ontology implementation and use contributions	Iker Esnaola-Gonzalez, Ignacio Lázaro – TEK	28/09/2021
0.6	IMP review	Dea Pujic -IMP	19/10/2021
0.7	FHG contribution	Marco Mittelsdorf - FHG	19/10/2021
0.8	TEK integration and new contribution based on comments	Iker Esnaola-Gonzalez, Ignacio Lázaro – TEK	05/11/2021

0.9	AIT review	Stefan Übermasser - AIT	12/11/2021
0.10	TEK review of suggestions from AIT's official review	Iker Esnaola-Gonzalez, Ignacio Lázaro - TEK	16/11/2021
0.11	IMP contribution to OpenADR	Lazar Berbakov - IMP	19/11/2021
0.12	TEK overall review	Iker Esnaola-Gonzalez - TEK	22/11/2021
1.0	Final Version	Ignacio Lázaro - TEK	24/11/2021

Executive Summary

This deliverable defines the REACT Semantic data model in the form of an ontology for the integration of all data interchanged between the equipment installed in Pilot sites and REACT cloud components, services, and tools. This document describes the process of the ontology definition and exploitation, thoroughly describing the methodology used, the implementation paying special attention to the ontology engineering best practices, the publication, and the instantiation.

This deliverable also defines the Open API to enable the exploitation of smart grid, and third-party services such as energy prices, weather forecasts, and data exchanging DR aggregators. A design of communication between REACT platform and smart grid by using OpenADR is provided.

Table of Contents

Executive Summary.....	5
1. Introduction.....	9
1.1 Scope	9
1.2 Audience.....	9
1.3 Abbreviations.....	9
1.4 Structure	10
2. The Semantic Web and Semantic Technologies	12
2.1 The data model.....	12
2.2 Linked Data	14
2.3 Ontologies.....	15
3. The REACT Ontology	17
3.1 Ontology Development Methodology	18
3.2 Ontology Implementation	21
3.2.1 Based on Ontology Design Patterns	22
3.2.2 Reusing well-known ontologies.....	24
3.2.3 Following a modular approach	25
3.3 Ontology Publication.....	26
3.4 Ontology Instantiation	28
3.4.1 Represented information in a nutshell.....	30
3.5 The Ontology in Use	34
4. External System Connectivity	37
4.1 Smart Grid Connectivity	38
4.2 Weather Service Integration	45
4.3 Open API Concept.....	49
5. Conclusions	52
References.....	54
Appendix I – REACT ORSD.....	57
Appendix II – REACT ontology in use example.....	59

List of Figures

Figure 1: An RDF graph example (source: W3.org).....	13
Figure 2: LOD Cloud as of May 2020.	15
Figure 3: Ontology types according to Guarino.....	17
Figure 4: LOT Methodology.	19
Figure 5: Main classes and properties of the REACT ontology.....	22
Figure 6: Main classes and properties of the reused ODPs.	23
Figure 7: Main classes and properties of the HPOnt ontology.....	26
Figure 8: REACT ontology documentation page excerpt.	28
Figure 9: Data Point List Excel file excerpt.	29
Figure 10: Representation of a house, equipment and measurements using the REACT ontology.....	30
Figure 11: Virtuoso SPARQL Query Editor.....	35
Figure 12: Results obtained with the Virtuoso SPARQL Query Editor.	35
Figure 13: Overview of OpenADR actors and roles.....	39
Figure 14: VTN sends events to VEN in push mode (OpenADR Alliance, 2015).....	40
Figure 15: Building blocks of the general oadrDistributeEvent.	40
Figure 16 OpenADR oadrDistributeEvent	43
Figure 17 OpenADR oadrCreateReport.....	45
Figure 18 Weatherbit.io data adapter	46
Figure 19 OAS general outline	50

List of Tables

Table 1: Information related to facility and equipment represented after the REACT ontology instantiation.....	31
Table 2: Information related to measurements represented after the REACT ontology instantiation	33
Table 3: List of REACT services and tools that may exploit the ontology instantiation data.	34
Table 4: Pricing signals (OpenADR Alliance, 2015)	41
Table 5: Signal for storage resources. XXX represents real, apparent and reactive versions of power or energy (OpenADR Alliance, 2015)	42



Table 6 Weatherbit.io API endpoints (\$KEY - API key used for authorization, \$LAT and \$LON - latitude and longitude for the selected pilot location) 47

1. Introduction

1.1 Scope

This deliverable defines the semantic data model, as an ontology, to have a common vocabulary for the raw signals coming from the assets installed in the pilot sites and with the platform system components, such as time-series database, relational database, and core services.

1.2 Audience

The intended audience for Deliverable 6.3 consists of members of the REACT Consortium and the Project Officer. This document is public, and it can be helpful for any other projects involved in research activities related to Renewable Energy Assets, any public organization, person, or entity which aims for the improvement of energy self-sufficiency of geographical islands, and people who might be interested in attaining a foundational understanding of concepts and principles behind Semantic Technologies.

Therefore, partners in charge of activities related to the implementation, integration, and deployment of analytic services and visualization tools within the REACT project are part of the target audience of the deliverable.

1.3 Abbreviations

A: Ampere

AAA: Anyone can say Anything about Any topic

ADR: Automated Demand Respond

API: Application Programming Interface

CQ: Competency Questions

ESS: Energy Storage System

DC: Direct Current

DR: Demand Response

HTTP: Hypertext Transfer Protocol

HP: Heat Pump

Hz: Hertz

IRI: Internationalized Resource Identifier

kW: Kilowatt

kWh: Kilowatt-hour

LOD: Linked Open Data

OAS: Open API Specification

OSRD: Ontology Requirements Specification Document

OWA: Open World Assumption

PV: Photovoltaic

REST: Representational State Transfer

RDF: Resource Description Framework

SPARQL: SPARQL Protocol and RDF Query Language

TSDB: Time Series Database

URI: Uniform Resource Identifier

V: Voltage

W3C: World Wide Web Consortium

wh: Watt-hour

1.4 Structure

- **Section 1:** it contains an overview of this document, providing its Scope, Audience, Abbreviations and Structure.
- **Section 2:** it provides a brief introduction to the Semantic Web and Semantic Technologies, as well as to related concepts such as Linked Data and ontologies.
- **Section 3:** it describes the REACT ontology, not only from a methodological and implementation point of view, but also from its instantiation and usage.

- **Section 4:** it contains the description regarding external connectivity, such as connectivity with smart grid, external weather service and open ADR concept.
- **Section 5:** a discussion and conclusions of the deliverable are provided.

The report includes two appendices:

- **Appendix I:** Contains an excerpt description of the ORSD of the Ontology.
- **Appendix II:** Contains the RDF model of an instantiation example of the REACT ontology.

2. The Semantic Web and Semantic Technologies

Nowadays, most Web content is suitable for human consumption, but it is not well supported by machines. This derived in the advent of the Semantic Web, which is not a separate Web but an extension in which information is given well-defined meaning, enabling computers and people to work in cooperation (Berners-Lee, Hendler, & Lassila, 2001). In fact, the Semantic Web builds upon the principles and technologies of the Web. It reuses the Web's global indexing and naming scheme, and Semantic Web documents can be accessed through standard Web browsers as well as through semantically aware applications (Domingue, Fensel, & Hendler, 2011). The World Wide Web was derived from a new way of thinking about sharing information. Therefore, it has a set of features that can be summarized as follows (Allemang & Hendler, 2011):

- The AAA (Anyone can say Anything about Any topic) slogan. In a web of documents, this slogan means that anyone can write a page saying whatever they want and publish it. In the case of the Semantic Web, AAA means that any individual can express a piece of data about some entity and this data can be combined with information from other sources.
- The Open World Assumption (OWA). Because of the AAA slogan, there could always be something new. Therefore, statements about knowledge that are not included or inferred from the explicitly recorded data may be considered unknown, rather than wrong or false.
- Non unique naming assumption. This feature is built upon the assumption that not all the contributors to the Web will coordinate with regards to the naming of entities. Therefore, the same entity could be referred to using more than one name.
- The network effect. This is the property thanks to which the value of joining in the Semantic Web increases with the number of people who have already joined, resulting in a spiral of participation.
- The data wilderness. The condition of the data that contains valuable information, but there is no guarantee that it will be readily understandable.

2.1 The data model

The Resource Description Framework (RDF) is a W3C (World Wide Web Consortium) recommendation for representing information on the Web¹. The basic structure are triples, which consist of a subject, a predicate, and an object. A set of RDF triples constitutes an RDF graph, which can be viewed as node and directed-arc diagrams. An example of an RDF graph is shown in Figure 1.

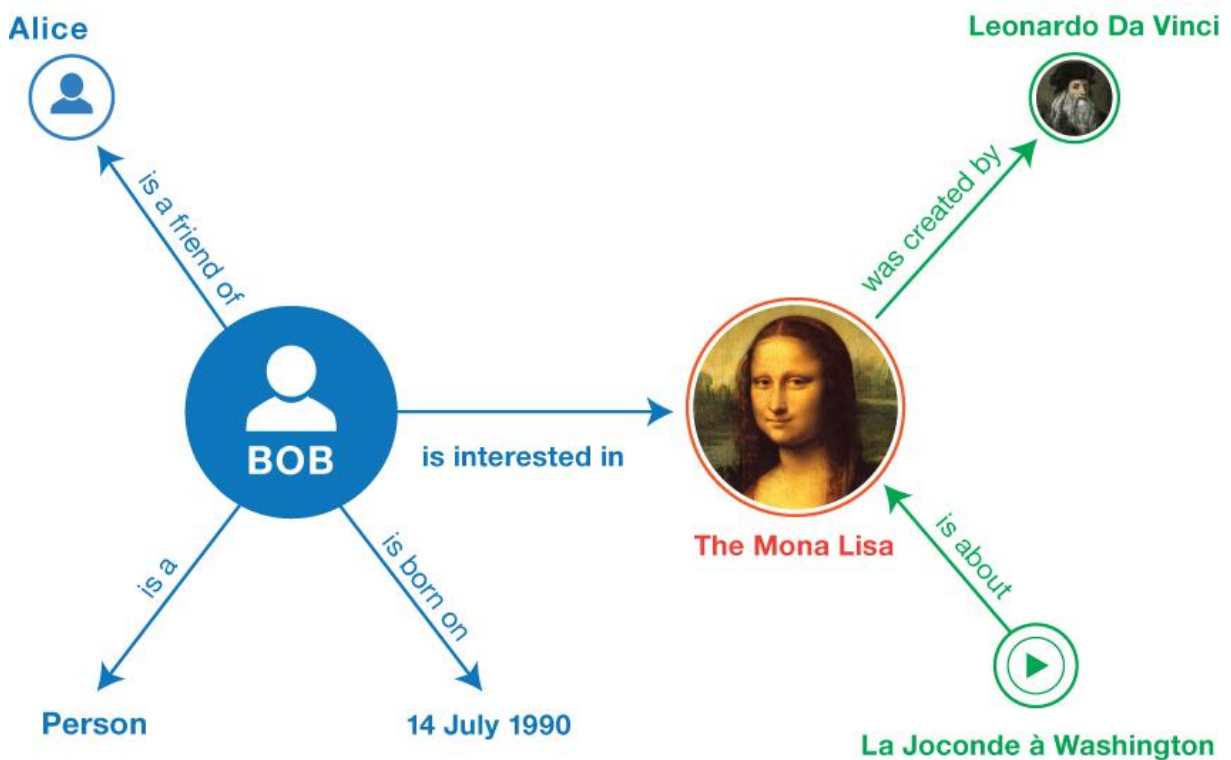


Figure 1: An RDF graph example (source: W3.org)

These resources are described using IRIs (Internationalized Resource Identifier). The IRI extends the ASCII characters subset of the URIs (Uniform Resource Identifier). Since a property is also an IRI, it can again be used as a resource interlinked to another resource. Furthermore, in RDF, IRIs can refer to anything. This flexibility makes the data model suitable for the context of an open Web.

It is important to note that RDF is not a data format, but a data model for describing resources as node-and-arc-labelled directed graphs. Therefore, although expressing RDF

¹ <http://www.w3.org/TR/rdf-primer/>

triples as a graph may be suitable to display data, this may not be the most compact or human-friendly way to see the relation between entities. These needs derived in different RDF serialization formats. RDF/XML and RDFa are standardized by the W3C, but there are many other more easily understandable non-standard serialization formats such as N-Triples and Turtle.

2.2 Linked Data

The term Linked Data (LD) refers to a set of best practices for publishing and interlinking structured data on the Web (Heath & Bizer, 2011). These best practices are also known as Linked Data principles², and they can be summarized as follows:

- Use URIs as names for things.
- Use HTTP URIs, so that people can look up those names.
- When someone looks up a URI, provide useful information, using the standards.
- Include links to other URIs so that they can discover more things.

To publish data on the Web, Linked Data uses HTTP URIs to identify the real-world items of a domain of interest. Other URI schemes such as URNs (Uniform Resource Name) and DOIs (Digital Object Identifier) are avoided, as HTTP URIs enable creating globally unique names in a decentralized way, and they serve as a means of accessing information describing the identified entity.

Any HTTP URI should be referenceable, which means that HTTP clients should be able to look up the URI and retrieve a description of the resource identified by such a URI. Furthermore, these descriptions should ideally be represented as HTML when they are intended to be read by humans, and as RDF data if intended consumers are machines. This can be achieved with an HTTP mechanism called content negotiation. This mechanism consists in HTTP clients sending HTTP headers with each request indicating which kind of documents they prefer. Afterwards, servers examine these headers and select the appropriate response.

² <https://www.w3.org/DesignIssues/LinkedData.html>

The LOD (Linked Open Data) Cloud³ presents datasets published in the Linked Data format. As of May 2020, the LOD cloud contained more than 1,300 datasets with 16,283 links, as it can be seen in Figure 2.

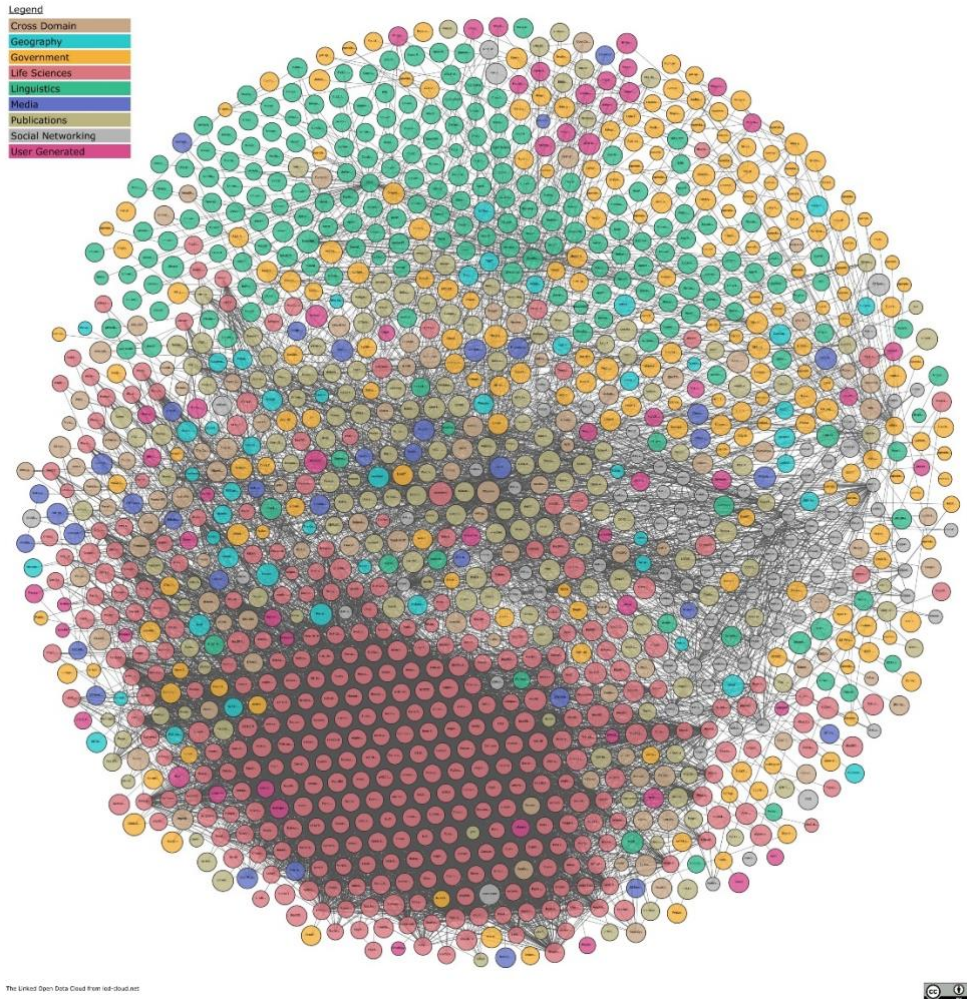


Figure 2: LOD Cloud as of May 2020.

2.3 Ontologies

The term ontology comes from the Greek *ontos* (being) and *logos* (word), and it was first used in philosophy in the nineteenth century to define the study of the nature of being, existence, or reality, as well as the basic categories of being and their relations (Breitman,

³ <https://lod-cloud.net/>

Casanova, & Truszkowski). In computer and information science field, an ontology can be understood as “a formal, explicit specification of a shared conceptualisation” (Studer, Benjamins, & Fensel, 1998). Therefore, ontologies appear to describe and represent a certain phenomenon, topic, or subject area through the description of classes, properties, and instances (also known as individuals).

Ontologies can be viewed as a spectrum of detail in their specification (Lassila & McGuinness, 2001). Catalogues, which consist of a finite list of terms used for expressing knowledge of information, are placed in the lowest end of the spectrum. This list of terms may not have descriptions at all, and their meaning can only be estimated because they are chosen from natural language. Likewise, there are no formal relations expressed between these terms. Therefore, often, such specifications are not referred to as ontologies. When at least one formal relation is defined and used between terms, the concept “ontology” can be used to refer to such a catalogue. From this point onward, there are languages that provide sets of constructs to describe an ontology, such as frames or simplified logic. As the specificity increases, the precision and the ability to use tools to automatically integrate systems also increases. However, the cost of building and maintaining a metadata registry increases accordingly.

Furthermore, ontologies can be categorised into different types according to their level of generality (Guarino, 1998), as it can be viewed in Figure 3:

- Top-level ontologies (often referred to as upper ontology or foundation ontology, general, or cross domain ontology) represent very general concepts which are independent of a specific domain or problem such as time, space, and events.
- Domain ontologies describe fundamental concepts according to a generic domain and specialise terms introduced in top-level ontology.
- Task ontologies describe fundamental concepts according to a general activity or task and specialise the terms introduced in top-level ontologies.
- Application ontologies are specialised ontologies focused on a specific task and domain. They are often a specialisation of both task and domain ontologies, and they also often specify roles played by domain entities for specific activity.

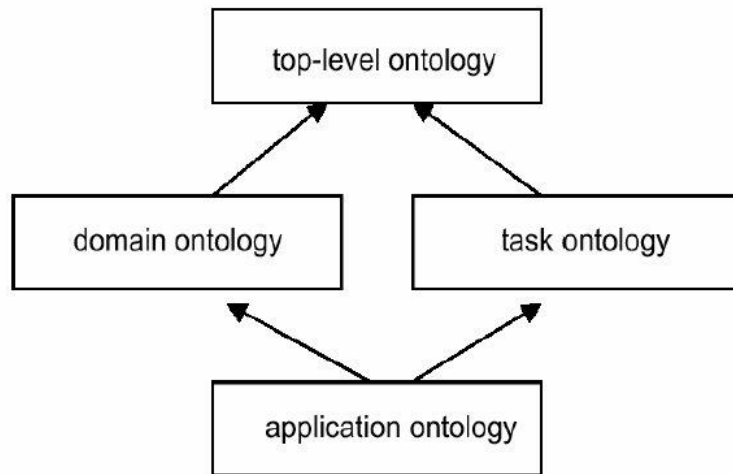


Figure 3: Ontology types according to Guarino.

Ontology-based approaches are proved to bring many advantages. Annotating raw data with terms coming from ontologies not only allows a better representation of the data itself, structuring it and setting formal types, relations, properties, and constraints that hold among them, but also enables representing data coming from multiple sources in a uniform way, thereby supporting data integration and data interoperability at a semantic level. Furthermore, additional background knowledge about a domain can be added to the set of available data with ontologies. This leads to the enrichment of the data set at hand, as well as enabling the application of indexing techniques to ensure the retrieval and navigation through related resources. Finally, after a semantic annotation process, data is more domain-oriented than the original source and allows more application-independent solutions. Consequently, there is no need for the user to be aware of the underlying structure of the raw data.

3. The REACT Ontology

The REACT ontology is aimed at supporting the Semantic Repository to enable the exploitation of the collected raw data's underlying semantics to provide other REACT services with the knowledge required to implement their functionalities. Furthermore, the representation of data based on ontologies not only will contribute to the interoperability of the solution at a semantic level but will also enable the sharing of unambiguous data within the different parties involved.

Deliverable D6.1 (REACT D6.1, s.f.) defines the system reference architecture of the REACT project where the platform components and relationships between them are shown. The Semantic Repository has a direct relationship with the historical data components, analytic services, and visualization tools. The Semantic Repository provides the analytical services and visualization tools with the required data to perform their functionalities.

The Semantic Repository stores the topology of the installations performed in pilot sites, defining the relevant metadata associated to the installation, the equipment installed, and the data gathered by them. It also defines the relationships among the equipment in a hierarchical structure and the specific data gathered by each equipment with the relevant attributes, some of them also related to the Canonical Data Model (CDM) defined in REACT platform as common message format for the successful syntactic interoperability of the data exchanged among the field and the cloud level components.

Furthermore, the Semantic Repository stores the required link to map the installations with the time-series database (TSDB), in charge of storing the raw data coming for the equipment at the field level.

3.1 Ontology Development Methodology

In order to ensure the final quality of an ontology, this must be carefully designed and implemented. Therefore, the use of well-founded ontology development methodologies is advised. After reviewing different methodologies including OnToKnowledge and DILIGENT (Pinto, Staab, & Tempich), the REACT ontology developers decided to follow the LOT (Linked Open Terms) Methodology⁴. LOT is an industrial method for developing ontologies and vocabularies, and it has been used in other H2020 funded projects such as VICINITY⁵, DELTA⁶ or BIMERR⁷. It can be considered an evolution of the NeOn methodology (Suárez-Figueroa, Gómez-Pérez, Motta, & Gangemi, 2012), as it lightens it with sprints iterations (from the scrum software development framework) and covers the ontology publication following the best practices. A summary of the LOT methodology is shown in Figure 4.

⁴ <https://lot.linkeddata.es/>

⁵ <https://www.vicinity2020.eu/>

⁶ <https://www.delta-h2020.eu/>

⁷ <https://bimerr.eu/>

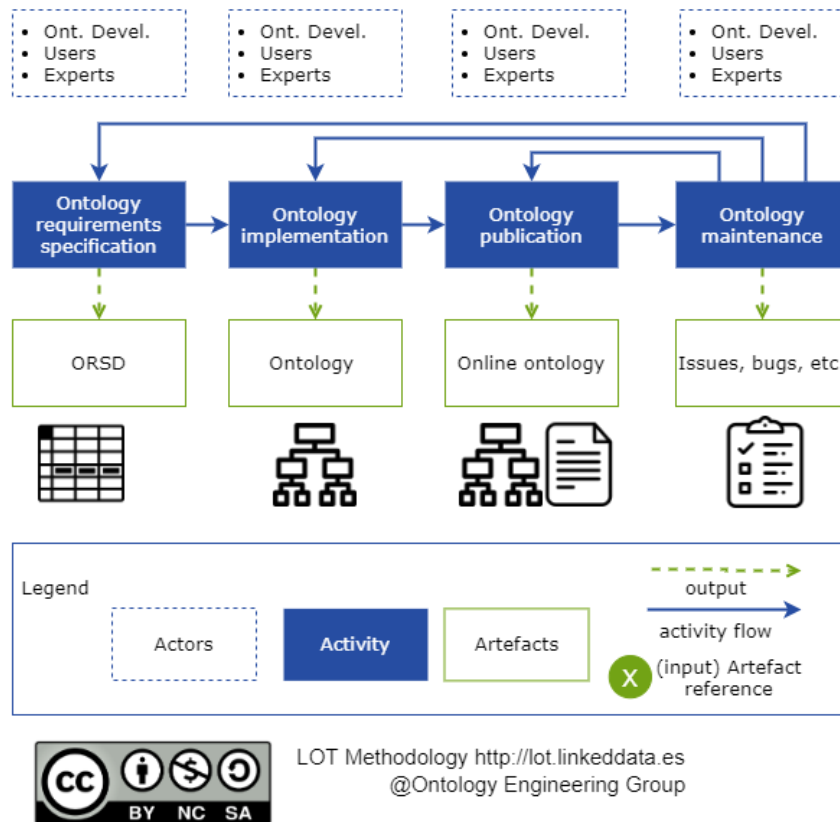


Figure 4: LOT Methodology.

The first stage of the LOT methodology, the requirements specification process, aims to state why the ontology is being built and to identify and define the requirements the ontology should fulfil. In the context of the REACT ontology, the ORSD (Ontology Requirements Specification Document) collected the ontology purpose, its intended users, and the set of functional requirements in the form of Competency Questions (CQ). It is worth mentioning that this information was retrieved from the different REACT stakeholders. An excerpt of the ORSD is shown in Appendix I – REACT ORSD.

The second stage deals with the actual implementation of the ontology. In this stage, ontology engineers need to formalize all the requirements collected in the previous stage. However, these requirements are first divided and categorized according to their priority, so that the implementation activity can be carried out iteratively. This means that, in each iteration, only a number of requirements are addressed, and a new ontology version is generated. This iterative approach makes less costly to make modifications in case of change of requirements, eases the testing of the ontology in early stages of development, and aids to better manage the potential risks.

According to (Simplerl, 2009), the reuse of ontological resources built by others and that have already reached some degree of consensus, is good practice in ontology development processes. Additionally, the W3C's Data on the Web Best practices⁸ states that reusing an existing ontology not only captures and facilitates consensus in communities, but it also increases interoperability and reduces redundancies. Furthermore, this practice brings other important benefits:

- It increases the quality of the applications reusing ontologies, as they become interoperable and they are provided with a deeper, machine-processable and commonly agreed-upon understanding of the underlying domain of interest.
- It reduces the costs related to ontology development because it avoids the reimplementation of ontological components which are already available on the Web and can be directly (or after some additional customisation tasks) integrated into the target ontology.
- It may improve the quality of the reused ontologies, as these are continuously revised and evaluated by various parties through reuse.

As any other task in the ontology development process, this ontology reuse should be approached in a methodological way. The Ontological Resource Reuse Process proposed by (Fernández-López, Suárez-Figueroa, & Gómez-Pérez, 2012) describes the set of activities to be performed for the reuse of existing ontological resources.

- **Ontology Search.** This activity consists in finding appropriate ontological resources that meet the requirements aimed to be satisfied. According to (Gyrard, Zimmermann, & Sheth, 2018), the existing ontology catalogues such as LOV⁹ or LOV4IoT¹⁰ (specialised in ontologies related to IoT) can ease this task.
- **Ontology Assessment.** This activity deals with assessing the usability of an ontology with respect to the requirements previously defined. However, this may end up being a laborious task due to the different criteria that may make ontologies suitable for a certain use case. Furthermore, the frequent scarce documentation of ontologies may hinder this activity as potential reusers may not understand the analysed ontology.

⁸ <https://www.w3.org/TR/dwbp>

⁹ <http://lov.linkeddata.es>

¹⁰ <https://lov4iot.appspot.com/>

- **Ontology Comparison.** In this activity, assessed ontologies should be compared according to criteria that encompass the content of the ontology, the organisation of these contents, the language in which it is implemented, the methodology that has been followed to develop it, the software tools used to build and edit the ontology, and the costs of the ontology as suggested by (Lozano-Tello & Gómez-Pérez, 2004).
- **Ontology Selection.** After assessing and comparing ontologies, the most appropriate one or ones must be selected and reused by integrating them in the new ontology being developed.

With a view to verify that each of the intermediate REACT ontology versions satisfy the ontology requirements identified in the ORSD, a validation process has been performed at the end of each iteration. This validation has been performed with Themis¹¹, a web-based tool which provides a set of test expressions based on lexico-syntactic patterns to check whether ontology requirements are satisfied. For each ontology version, a set of tests have been designed, implemented, and run to verify that the targeted CQs are adequately addressed, and the desired knowledge is modelled. These tests have been represented with the Verification Test Case (VTC) ontology¹² and exported in RDF files, to run them in the future when the REACT ontology may be modified.

Finally, The REACT ontology's design correctness was evaluated with OOPS! (Ontology Pitfall Scanner). OOPS! is an online tool¹³ and detects some of the most common pitfalls appearing within ontology developments. Namely, it evaluates an ontology against a catalogue of 41 potential pitfalls classified into three levels according to their severity: minor, important and critical. The use of this tool contributed to an early detection of pitfalls and complemented the manual review of the ontology's correctness.

3.2 Ontology Implementation

As mentioned in previous sections, the REACT ontology has been developed following the LOT methodology to ensure its high-quality. Furthermore, additional ontology quality aspects suggested in (Esnaola-Gonzalez, Bermúdez, Fernandez, & Arnaiz, 2020) have been

¹¹ <http://themis.linkeddata.es>

¹² <https://w3id.org/def/vtc>

¹³ <http://oops.linkeddata.es/>

considered to improve the potential reusability of the ontology. The main classes and relationships of the REACT ontology can be seen in Figure 5.

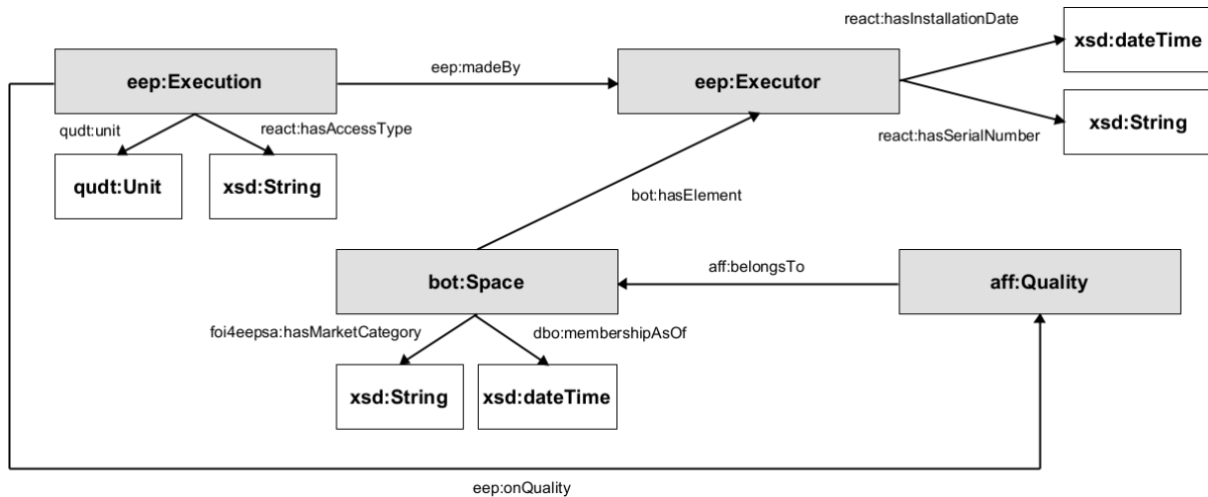


Figure 5: Main classes and properties of the REACT ontology.

3.2.1 Based on Ontology Design Patterns

The use and combination of Ontology Design Patterns (ODP) is conceived as a suitable option when developing ontologies, due to the great flexibility provided which allows a proper segmentation of the intended conceptualisation (Hitzler, Gangemi, & Janowicz, 2016). And this is the way in which the REACT ontology has been designed. However, instead of defining our own ODPs, following the ontology engineering best practices, a thorough analysis of existing ODPs have been performed searching in specialised catalogues such as [OntologyDesignPatterns.org](http://ontologydesignpatterns.org/)¹⁴, in order to reuse existing ones. Namely, the two ODPs reused have been: the AffectedBy ODP¹⁵ and the Execution-Executor-Procedure (EEP) ODP¹⁶. It is worth mentioning that these ODPs have already been reused by different ontologies such as for supporting a data analyst assistant in energy efficiency and thermal comfort problems in buildings (Esnaola-Gonzalez, Bermúdez, Fernandez, & Arnaiz, EEPSA as a core ontology for energy efficiency and thermal comfort in buildings, 2021) or to cover the agri-food domain (Esnaola, y otros, 2019). Figure 6 shows the main classes and properties of the mentioned ODPs.

¹⁴ <http://ontologydesignpatterns.org/>

¹⁵ <https://w3id.org/affectedBy>

¹⁶ <https://w3id.org/eep>

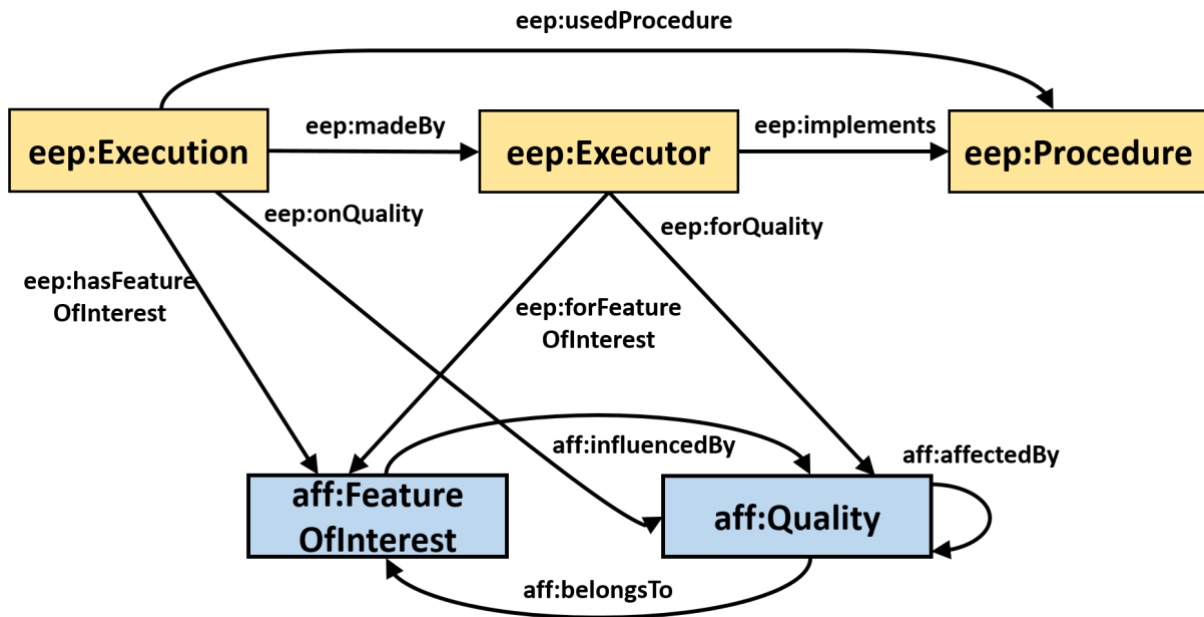


Figure 6: Main classes and properties of the reused ODPs.

The AffectedBy ODP defines two classes representing features of interest (*aff:FeatureOfInterest*) and their qualities (*aff:Quality*) and three object properties: *aff:belongsTo*, *aff:affectedBy* and *aff:influencedBy*. The *aff:belongsTo* object property supports the notion that every quality belongs to the feature of interest it is intrinsic to (i.e. a quality cannot belong to different features of interest), thus following the conceptualisation defined in the DOLCE upper level ontology (Borgo & Masolo, 2009). The *aff:affectedBy* object property relates a quality with another quality that it affects, and the *aff:influencedBy* object property relates a quality with the feature of interest that it influences.

The EEP ODP imports the AffectedBy ODP and its two classes, and additionally, it defines three more classes: *eep:Execution*, *eep:Executor*, and *eep:Procedure*. An individual of *eep:Execution* is an event (e.g. a forecast) upon a quality of a feature of interest, produced by an agent by performing a procedure. As for an individual of *eep:Executor*, it is an agent capable of performing tasks by following procedures. Lastly, an individual of *eep:Procedure* describes the workflow, protocol, plan, algorithm, or computational method to be executed by agents to produce an event. Furthermore, the *eep:madeBy*, *eep:usedProcedure*, and *eep:onQuality* object properties are introduced in the EEP ODP. The *eep:madeBy* object property links an execution to the agent that performs the action, the *eep:usedProcedure* object property links an execution to the procedure that describes the task to be performed; and the *eep:onQuality* object property links an execution to the

quality concerned by the execution. These three functional object properties, combined with a set of property chain axioms defined in the EEP ODP, allow the inference of the remaining object properties *eep:implements* linking executors to procedures, *eep:hasFeatureOfInterest* linking executions to features of interest, *eep:forQuality* linking executors to qualities, and *eep:forFeatureOfInterest* linking executors to features of interest.

3.2.2 Reusing well-known ontologies

The ODPs mentioned are conceived as the backbone of the REACT ontology. On top of these ODPs, a set of classes, individuals, relationships and restrictions are constructed to satisfy all the requirements previously indicated. Once again, instead of directly encoding our own concepts, the REACT ontology's development has been based on the best practices and an ontology reusing effort has been made. Following the previously defined steps, different ontologies have been searched, compared, evaluated, and finally selected according to their suitability for satisfying the collected platform and user requirements. Some of the selected and (partially) reused ontologies are the following ones:

- SAREF (Smart Appliances REference) ontology¹⁷ facilitates the matching of existing assets in the smart appliances' domain. The central concept of the ontology is the *saref:Device* class, which is modelled in terms of functions, associated commands, states and provided services. The REACT ontology has reused properties such as *saref:hasManufacturer* to represent the manufacturer of a given equipment.
- BOT (Building Topology Ontology)¹⁸ is a minimal ontology developed by the W3C LBD (Linked Building Data) Community Group for covering core concepts of a building and for defining the relationships between their subcomponents. BOT serves as an ontology that could promote its reuse as a central ontology in the AEC (Architecture, Engineering and Construction) domain. In the context of the REACT ontology, classes such as *bot:Site* and *bot:Space* have been reused for representing the basic topological information, and properties such as *bot:hasElement* to represent the relationship between the equipment contained within the pilot sites.
- The QUDT¹⁹ is an initiative sponsored by the NASA to formalize Quantities, Units of Measure, Dimensions and Types using ontologies. In this regard, the REACT

¹⁷ <https://w3id.org/saref>

¹⁸ <https://w3id.org/bot>

¹⁹ <http://www.qudt.org>

ontology has leveraged certain properties such as *qudt:unit* to associate observations/actuators with the units in which they are measured, and additionally, certain classes from the UNIT ontology (which is also part of the QUDT) like *unit:W-HR* for Wh and *unit:DEG_C* for Degrees Celsius have been reused to represent such units of measurements.

3.2.3 Following a modular approach

The ontology modularization consists in partitioning them into independent self-contained knowledge components. Such a modular approach brings benefits, including the flexibility for component reuse (Grau, Horrocks, Kazakov, & Sattler, 2008), the support for more efficient query answering (Stuckenschmidt & Klein, 2007), and the enhancement of component changes and evolution (Ensan & Du, 2013).

When an already existing ontology is large and monolithic, it needs to be split up in order to benefit from the mentioned advantages. There are different techniques that perform ontology partitioning by dividing an ontology into a set of significant modules that together form the original ontology. However, there is no universal way to modularize an ontology (d'Aquin, Schlicht, Stuckenschmidt, & Sabou, 2009), and the choice of a particular technique or approach should be guided by the requirements of the application or use case.

The implementation of ontology modularization techniques is advised in early ontology development stages because, otherwise, it could end up being a complex task. This is why, for the development of the REACT ontology, it has been considered from the very beginning. During the ontology requirements collection, those pertaining the heat pump systems topic were grouped and separated from the rest with a view to developing the HPOnt (Heat Pump Ontology) to benefit from the aforementioned characteristics. The HPOnt aims to formalize and represent all the relevant information of Heat Pump systems such as their cooling capacity (*hpont:hasNominalCoolingCapacity*) or consumption in heating mode (*hpont:hasNominalPowerConsumptionInHeatingMode*). Figure 7 shows the main properties of HPOnt.

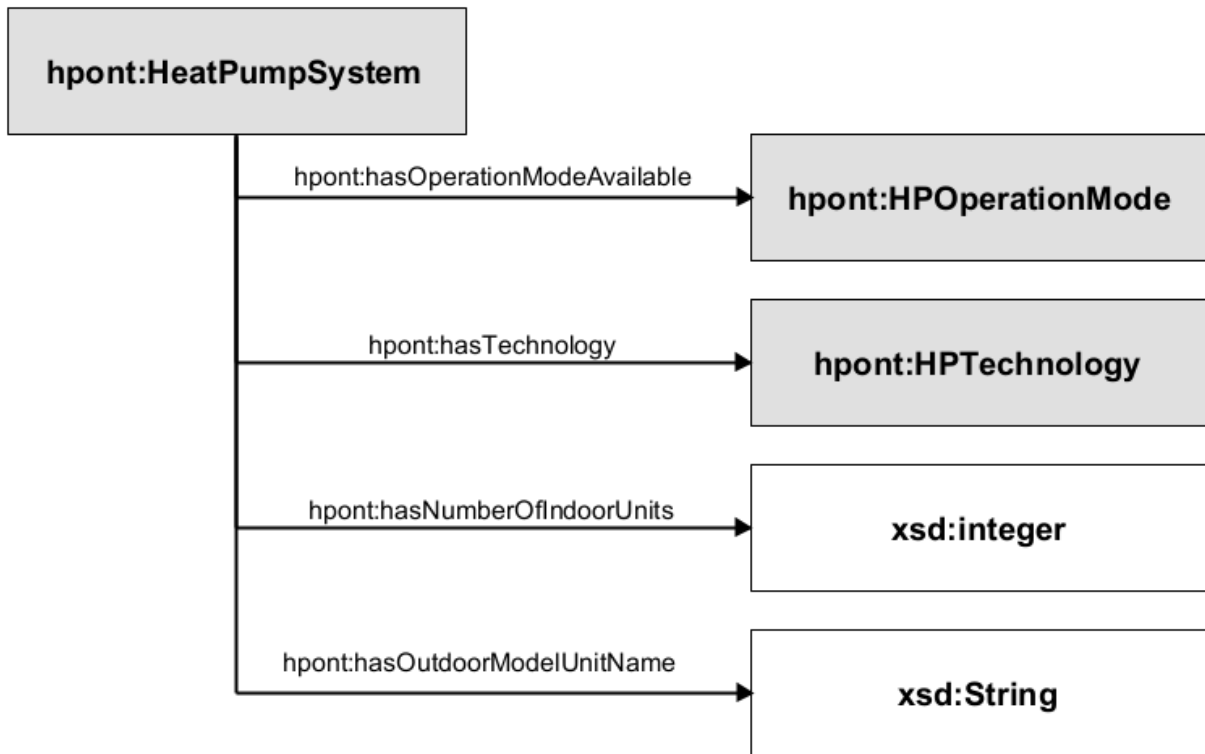


Figure 7: Main classes and properties of the HP ontology.

3.3 Ontology Publication

A good ontology documentation increases its understandability and potential usability, both by experts in semantics and by people who are not necessarily experts (Peroni, Shotton, & Vitali, 2013). The documentation of the REACT ontology is generated with WIDOCO (a Wizard for DOCUMENTING Ontologies) developed by (Garijo, 2017) which creates a set of linked enriched HTML pages. These HTML pages are extended with hand-made sections such as the alignments to other ontologies or with ontology usage examples.

For URL stability and manageability purposes, the W3C Permanent Identifier Community Group's²⁰ w3id.org²¹ redirection service is used. The purpose of this initiative is to provide a secure, permanent URL re-direction service for Web applications.

²⁰ <https://www.w3.org/community/perma-id/>

²¹ <https://w3id.org/>

Furthermore, W3C's Data on the Web Best Practices²² states that providing metadata is a fundamental requirement that helps human users and computer applications to understand the data as well as other important aspects that describe a dataset. All the ontological resources presented in this article are annotated following guidelines described by Garijo and Poveda-Villalón²³ as the REACT partners involved in the task T6.3 consider the most complete guideline among the ones reviewed. As a matter of fact, both the ontology itself and the classes and properties are annotated with all the recommended terms as well as some additional optional terms.

Figure 8 shows an excerpt of the REACT ontology documentation available in <https://w3id.org/react>. Likewise, the HPOnt ontology imported by the REACT ontology, is available online in <https://w3id.org/hpont>.

²² <https://www.w3.org/TR/dwbp/>

²³ <https://w3id.org/widoco/bestPractices>

The REACT Ontology

Release: 2021-06-01

Last Update: 2021-09-14

This version:

<https://w3id.org/react>

Revision:

0.3

Authors:

Iker Esnaola-Gonzalez, Tekniker <iker.esnaola@tekniker.es>

Contributors:

Ignacio Lazaro, Tekniker <ignacio.lazaro@tekniker.es>

Nikola Tomasevic, Institut Mihajlo Pupin <nikola.tomasevic@pupin.rs>

Marko Jelic, Institut Mihajlo Pupin <marko.jelic@pupin.rs>

Dea Pujic, Institut Mihajlo Pupin <dea.pujic@pupin.rs>

Lluis Millet, Fraunhofer-Institut für Solare Energiesysteme <lluis.millet.biosca@ise.fraunhofer.de>

Imported Ontologies:

[BOT \(Building Topology Ontology\)](#)

[EEP \(Execution-Executor-Procedure ODP\)](#)

[HPOnt \(The Heat Pump Ontology\)](#)

Download serialization:

Format [JSON LD](#)

Format [RDF/XML](#)

Format [N Triples](#)

Format [TTL](#)

License:

License <https://creativecommons.org/licenses/by-nc/4.0/>

Visualization:

Visualize with [WebVowl](#)

Figure 8: REACT ontology documentation page excerpt.

3.4 Ontology Instantiation

The main aim of the ontology is to adequately represent the information of the different pilot sites regarding the different equipment installed, as well as the inherent features of the facilities involved in the pilot site. This information is collected by pilot case coordinators in the form of Excel files called Data Point Lists, as shown in Figure 9.

Unique identifier of the facility where the equipment is installed	Category of the facility	Date when the facility installation was finished	Energy carriers present in a given facility as local generators	Energy carriers present in a given facility as demand	Topology of the facility
Free input	Select from the list	Free input	Free input	Free input	Select from the list
Location_facility_id	Facility_category	Facility_membership_date	Generation_energy_carriers	Demand_energy_carriers	Facility_topology
LG4	residential	02/12/2020	electric	electric	DC Coupled
LG11	residential	02/12/2020	electric	electric	DC Coupled
LG27	residential	02/12/2020	electric	electric	DC Coupled
LG18	residential	16/06/2021	electric	electric	DC Coupled
LG2.1	residential	02/12/2020	electric	electric	DC Coupled

Figure 9: Data Point List Excel file excerpt.

The transformation from the data collected in Data Point Lists to the ontology instantiation (the so-called A-Box) has been automated with a service based on Apache Jena framework²⁴. Apache Jena is a free and open-source Java framework for building Semantic Web and Linked Data applications. The developed service enables the extraction of the information from the Excel sheets, its semantic annotation with appropriate ontology terms, and its storage to an RDF Store where it will remain accessible.

As for selecting the RDF Store for storing this information, two criteria have been considered. On the one hand, the ranking of popular RDF Stores provided by specialized webpages such as db-engines²⁵, and on the other, the experience of project partners. After making a thorough analysis, the selected RDF Store has been Openlink Virtuoso²⁶, namely the OpenLink Virtuoso Universal Server Version 07.20. As a matter of fact, apart from being successfully used in previous projects such as RESPOND H2020, Virtuoso is the second most popular RDF Store as of September 2021.

It is worth noting that the representation of measurements and actuations made by IoT systems and their storage in RDF Stores is not advisable, due to their poor performance. Instead, since this kind of data is characterized by its abundance, it is more suitable to be stored in a time series database, which are optimized for time-series data, thus being able to manage such an amount of data while ensuring a high performance. Therefore, the semantic representation of a given equipment in charge of performing an observation/actuation will have the same identifier both in the semantic repository and in the TSDB (e.g., <https://react2020.eu/device/VIC-GXHQ2022T1Y6A-100>).

²⁴ <https://jena.apache.org/>

²⁵ <https://db-engines.com/>

²⁶ <https://virtuoso.openlinksw.com/>

Figure 10 shows a representation of a given house, its equipment and measurements using the REACT ontology terms. The corresponding RDF model in Turtle notation is provided in Appendix II – REACT ontology in use example.

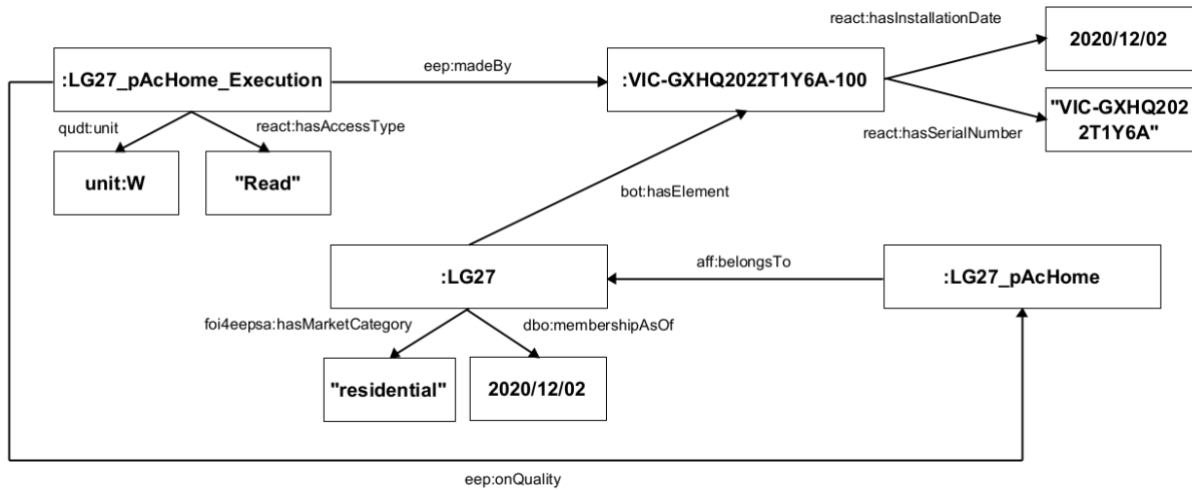


Figure 10: Representation of a house, equipment and measurements using the REACT ontology.

3.4.1 Represented information in a nutshell

After performing the REACT ontology instantiation process, the target RDF Store contains, on the one hand, information related to the facilities and equipment installed, and on the other hand, the list of the energy-related measurements collected by such equipment, whose values will be published and stored in the TSDB.

This data is key input for the implementation and execution at cloud level of the REACT platform services such as Energy Demand Forecast, PV Forecast, Optimization Service, Translation to Control Actions of the optimal profile, execution the automatic control of the batteries and heat pump (automated DR), as well to provide useful recommendations to the users for manual actions.

Table 1 includes a set of relevant pieces of information related to facility and equipment installed and represented after the REACT ontology instantiation.

Target concept	Parameter	Description
Facility	Max grid import	Maximum power draw from grid, technical limitation due to circuit breaker setup or contracted power

Facility	Max grid export	Maximum feed in power, technical limitation due to inverter setup or contracted power
Storage	Technology	Technology of the storage system installed, including Lithium Iron Phosphate (LFP), Hydrogen (H ₂), Sodium Nickel Chloride (NaNiCl ₂), Lithium Titanate (LTO), Pure Lead Carbon (PLH+C), or Lead-Acid (Pb)
Storage	Storage total capacity	Total storage capacity of the storage system (in kWh)
Storage	Max charge power	Maximum charging power (kW)
Storage	Max discharge power	Maximum discharging power (kW)
PV Panel	Technology	Technology of the Photovoltaic panels
PV Panel	Production total capacity	Total production capacity of the PV installation (in kWh)
PV Panel	Curtailable	Possibility of curtailment of PV production
Heat Pump	Technology	Technology of the Heat Pumps
Heat Pump	Power supply type	Electric power supply type of the HP system
Heat Pump	Operating mode	Operating mode available for the HP system
Heat Pump	Storage volume	Buffer storage volume of the HP system
Heat Pump	DHW storage volume	Domestic Hot Water storage volume of the HP system
Heat Pump	Nominal Cooling Capacity	Nominal Cooling Capacity of the HP system
Heat Pump	Nominal Heating Capacity	Nominal Heating Capacity of the HP system
Heat Pump	Nominal power consumption cooling	Nominal power consumption in cooling mode
Heat Pump	Nominal power consumption heating	Nominal power consumption in heating mode

Table 1: Information related to facility and equipment represented after the REACT ontology instantiation

Table 2 includes a subset of measurements represented after the REACT ontology instantiation. As it has been previously mentioned, the values of these measurements are not stored in the RDF Store but in the TSDB to ensure a high-performance.

Equipment	Measurement	Description
Facility	eAcGridIn	Grid Energy from net (wh)
Facility	eAcGridOut	Grid Energy to net (wh)
Facility	vAcGrid	Grid Voltage (V)
Facility	iAcGrid	Grid Current (A)
Facility	fAcGrid	Grid Frequency (Hz)
Facility	vAcHome	Home Voltage (V)
Facility	iAcHome	Home Current (A)
Facility	fAcHome	Home Frequency (Hz)
PV Panel	emppt	Accumulated Energy Produced by PV panels (wh)
PV Panel	yield	Energy Produced by PV panels (wh)
PV Panel	yieldYesterday	Energy Produced by PV panels in previous day (wh)
PV Panel	yieldPower	Power Producing by PV panels during day (w)
PV Panel	maxProductionPower	Maximum charge power today
PV Panel	maxProductionPower Yesterday	Maximum charge power yesterday
ESS	acPowerSetPoint	ESS control loop setpoint

ESS	relay1State	CCGX Relay 1 state
ESS	relay2State	CCGX Relay 2 state
Storage	vBat	Battery voltage (V DC)
Storage	iBat	Current (A DC)
Storage	tBat	Battery temperature (Degrees celsius)
Storage	sOc	State of charge (%)
Storage	nCycles	Charge cycles
Storage	sOh	State of health (%)
Storage	stateBat	Battery state System
Storage	eInBat	Charged Energy
Storage	eOutBat	Discharged Energy

Table 2: Information related to measurements represented after the REACT ontology instantiation

Having such a fine-grained representation of the data related to the different facilities participating in the REACT project, the set of potential REACT services that could benefit from the REACT ontology's instantiation is considerable. Some of this data might be directly used by these services such as the total facility production capacity by the REACT Energy Production Forecasting service, while other data may be complemented, for example, the gateway identifier to deliver request messages of automated control actions by the Battery Energy Control Dispatching service. Table 3 summarizes some of the services and tools that may exploit the REACT ontology instantiation.

Service/Tools	Relevant Data
Energy Demand Forecasting Service	Energy consumed in the building for training models

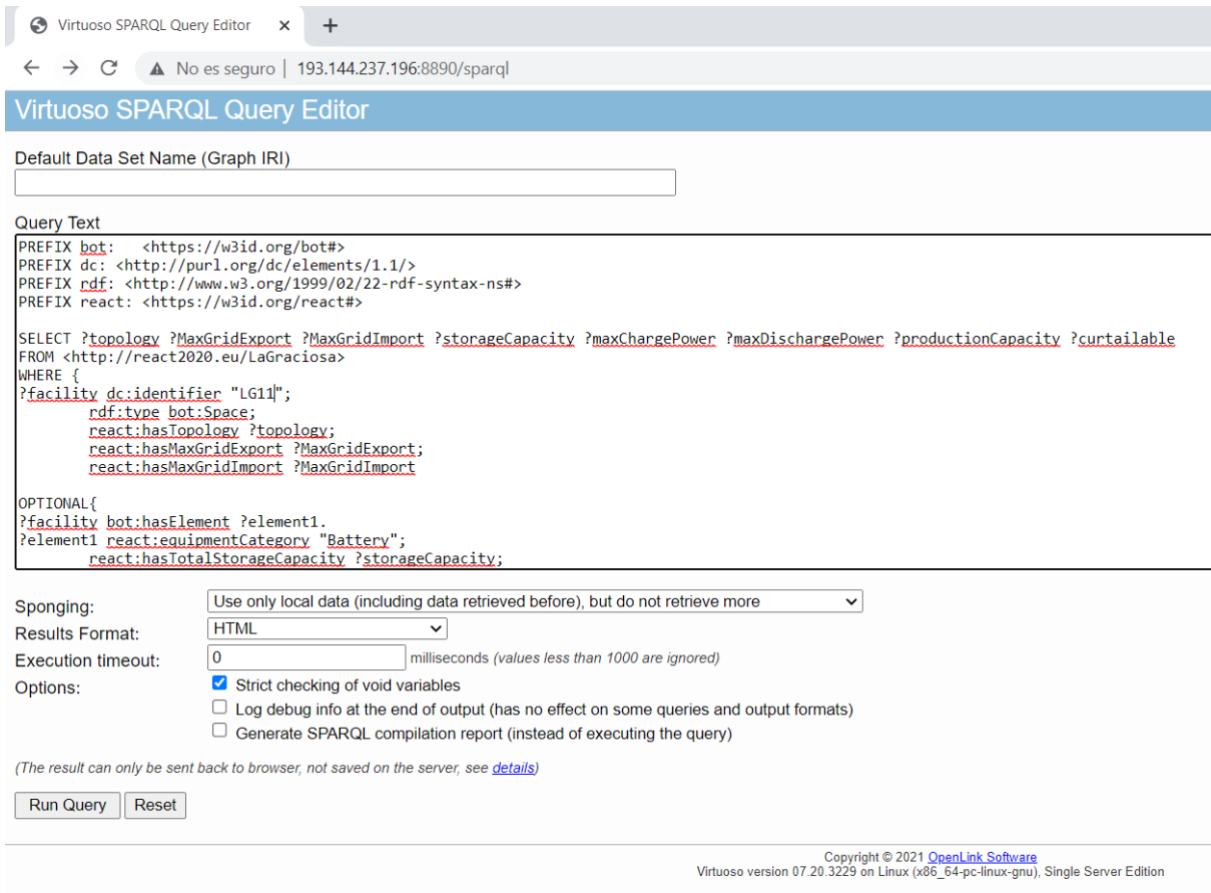
Energy Production Forecasting Service	PV Production total capacity for scalability of the PV forecast for each building
Optimization Service	Max grid import, Max grid export
Battery Energy Control Dispatching	Storage total capacity, Max charge power, Max discharge power . Gateway Id in charge of the control actions execution
Thermal Building Models	Technical characteristics of Heat Pumps. Gateway Id in charge of the control actions execution
Web & mobile app	Equipment of the installations, measurements monitored

Table 3: List of REACT services and tools that may exploit the ontology instantiation data.

3.5 The Ontology in Use

SPARQL (SPARQL Protocol and RDF Query Language) is a query language which can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. It is a W3C recommendation as of 2008 and enables querying information that can be RDF graphs or results sets. The syntax of a SPARQL query is similar to the SQL query syntax, as both of them use keywords such as SELECT to determine which subset of the selected data is returned, and WHERE to define graph patterns to find. In the context of the project, SPARQL queries are envisioned as the way to retrieve the information stored in the Semantic Repository.

The Competency Questions defined in the ORSD are the base for the definition of the queries required by the different services and components of the REACT platform. As commented earlier, Virtuoso is the base RDF Store of the Semantic Repository, which provides an online query tool called Virtuoso SPARQL Query Editor (shown in Figure 11) to execute SPARQL queries and get the results directly in an Internet Browser (shown in Figure 12).



Default Data Set Name (Graph IRI)

Query Text

```

PREFIX bot: <https://w3id.org/bot#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX react: <https://w3id.org/react#>

SELECT ?topology ?MaxGridExport ?MaxGridImport ?storageCapacity ?maxChargePower ?maxDischargePower ?productionCapacity ?curtailable
FROM <http://react2020.eu/LaGraciosa>
WHERE {
  ?facility dc:identifier "LG11";
    rdf:type bot:Space;
    react:hasTopology ?topology;
    react:hasMaxGridExport ?MaxGridExport;
    react:hasMaxGridImport ?MaxGridImport
}

OPTIONAL{
  ?facility bot:hasElement ?element1.
  ?element1 react:equipmentCategory "Battery";
    react:hasTotalStorageCapacity ?storageCapacity;
  }
  
```

Sponging: Use only local data (including data retrieved before), but do not retrieve more

Results Format: HTML

Execution timeout: 0 milliseconds (values less than 1000 are ignored)

Options:

- Strict checking of void variables
- Log debug info at the end of output (has no effect on some queries and output formats)
- Generate SPARQL compilation report (instead of executing the query)

(The result can only be sent back to browser, not saved on the server, see [details](#))

Run Query Reset

Copyright © 2021 [OpenLink Software](#)
 Virtuoso version 07.20.3229 on Linux (x86_64-pc-linux-gnu), Single Server Edition

Figure 11: Virtuoso SPARQL Query Editor.

topology	MaxGridExport	MaxGridImport	storageCapacity	maxChargePower	maxDischargePower	productionCapacity	curtailable
"AC Coupled"	"70 kW"	"70 kW"	"16.8 kWh"	"16.8"	"8.4 kW"	"16.8 kW"	true

Figure 12: Results obtained with the Virtuoso SPARQL Query Editor.

The use of the Virtuoso SPARQL Query Editor is a manual procedure to perform specific requests to the Semantic Repository, but it is not the adequate when these requests need to be automated for their integration with backend services and tools. Apart from the default query tool provided by Virtuoso, it is possible to use a program written in a specific programming language and with the required libraries, to access the Semantic repository and execute the corresponding SPARQL queries. And this is the approach followed in the REACT project.

The REACT Visualization Tools that require data stored in the Semantic Repository, use API methods to get the required data from the Semantic repository. These API methods are implemented in Java and, with the support of Apache Jena libraries, execute the desired SPARQL queries against the Virtuoso server. These API methods can be executed by a HTTP request (e.g., <https://react2020.eu/apiv0.1/info/semantic/pilot/LaGraciosa/es>) and the result is a JSON payload (see Listing 1) with the information provided by the specific SPARQL query.

```
{
  "id": "LG11",
  "productionCapacity": "16.8 kW",
  "storageCapacity": "16.8 kWh",
  "topology": "AC Coupled",
  "maxGridExport": "70 kW",
  "maxGridImport": "70 kW",
  "maxChargePower": "16.8 kW",
  "maxDischargePower": "8.4 kW",
  "curtailable": "true",
  (...)
}
```

Listing 1: SPARQL results by API HTTP request.

The defined SPARQL queries are common to the three REACT pilot sites, thus they are parameterizable to avoid unnecessary repetitions. Each SPARQL query has its own parameters and setting the adequate values for each of them allows retrieving the desired information regarding any specific facility, device, or observed quality. The results from a specific SPARQL will be used, for example, by the Visualization tools to get the raw data stored in the time-series database and plot the data in a graph. Let us consider a specific use case to illustrate the ontology exploitation process.

The Competency Question CQ02 has been defined in the ORSD as follows: *“Which are the parameters needed to get the values of a given measurement within a given house?”*. The input parameters this SPARQL query are *location_facility_id* (the identifier of the facility sought) and *measurement_id* (the identifier of the observed quality sought). The output values of this SPARQL query are *device_id* (the identifier of the device in charge of making the observation), and *measurement_index* (an index of the measurement, 1 for a scalar value, 1,2, or 3 for 3-phase measurement).

Considering that a given service needs to retrieve the state of charge of the battery for a specific house in La Graciosa, the CQ02 has been instantiated as shown in Listing 2. The

parameter *location_facility_id* has been set to “LG11” (the identifier of the facility at hand) and *measurement_id* to “sOc” (the identifier of the state of charge of the battery).

```
PREFIX aff: <https://w3id.org/affectedBy#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX eep: <https://w3id.org/eep#>
PREFIX react: <https://w3id.org/react#>

SELECT ?deviceID ?measurementIndex
FROM <http://react2020.eu/LaGraciosa>
WHERE {
  ?execution eep:madeBy ?executor;
    react:hasMeasurementIndex ?measurementIndex;
    eep:onQuality ?quality.

  ?executor dc:identifier ?deviceID.

  ?quality dc:identifier "sOc";
    aff:belongsTo ?FoI.

  ?FoI dc:identifier "LG_11".
}
```

Listing 2: SPARQL Query CQ02 parametrized for retrieving the state of charge of the battery of house LG11.

The output values obtained after executing this SPARQL query are “VIC-GXHQ2022UK7WV-225” (as the device identifier) and “1” (as the measurement index). With this information, the measurements registered in the Time-Series DB can be retrieved and afterwards displayed, by executing the parametrizable InfluxQL query shown in Listing 3.

```
SELECT (value) AS value FROM sOc WHERE (deviceId = 'https://react2020.eu/device/VIC-GXHQ2022UK7WV-225') AND measurementIndex = '1' AND time >= '2020-09-22T08:50:37Z' AND time < '2020-09-22T10:09:37Z'
```

Listing 3: InfluxQL Query CQ02 parametrized for retrieving the state of charge of the battery of house LG11.

4. External System Connectivity

This chapter describes the smart grid connectivity tools and services developed in REACT platform, in one hand, for the exploitation of data provided by third party services, such

as Weather data, and in other hand to provide access to the REACT core services by an Open API, and the integration with grid operators by the well-known OpenADR standard interface.

4.1 Smart Grid Connectivity

Moving towards a grid with a large number of distributed energy resources (DER), challenges arise to ensure grid stability and reliability. Grid operators and utilities have several options to match demand and supply as well as to avoid grid congestion. For instance, DERs could be curtailed during critical times of high production. Same applies to flexible consumers, which could be limited in their consumption or could be turned off. Apart from direct control, dynamic price signals could also be used as incentives to consume less energy on higher prices or vice versa.

All options require a communication channel from grid operator or utility to the DERs and flexible consumers. However, a smart grid often features a large diversity of software and hardware systems which need to exchange information with each other. Therefore, the core requirement for control and demand response actions is the interoperability between the systems. This can be achieved by utilizing open and widely used communications standards like OpenADR, IEC 61850, IEC 60870, OSCP.

This section investigates how grid operators or utilities could interact with the REACT cloud platform. This would enable the REACT cloud platform and its services to integrate requirements from the current grid state into control strategies and optimisation.

OpenADR has its origins in North America back in 2002 and was developed to enable automated demand response actions at customer facilities to improve grid reliability and economics. It supports demand response events for load shedding and shifting, as well as continuous dynamic price signals for hourly day-ahead or day-of real time pricing. In 2018, OpenADR has become an international standard, which is also known as IEC 62746-10-1:2018²⁷. Nowadays, OpenADR is used by grid operators and utilities around the globe for automated demand response actions (OpenADR Alliance, 2015).

The OpenADR architecture has two primary actors called Virtual Top Nodes (VTN) and Virtual End Nodes (VEN). The role of the VTN is usually played by a utility that manages the resources, creates DR events and requests reports from the resources. The VEN is

²⁷ <https://webstore.iec.ch/publication/26267>

typically a building or facility in commercial, industrial or residential domain, which features controllable resources. The VEN directly controls the resources based on the events of the VTN and generates reports for the VTN. Apart from this clear separation of VTN and VEN, there also actors like aggregators, which can play both roles at the same time. In the context of the REACT project, the REACT cloud platform would play the role of a VEN as depicted in Figure 13. Connected demand side resources (DSR) could be any controllable loads like storage systems, heat pumps or electric vehicles and their charging infrastructure.

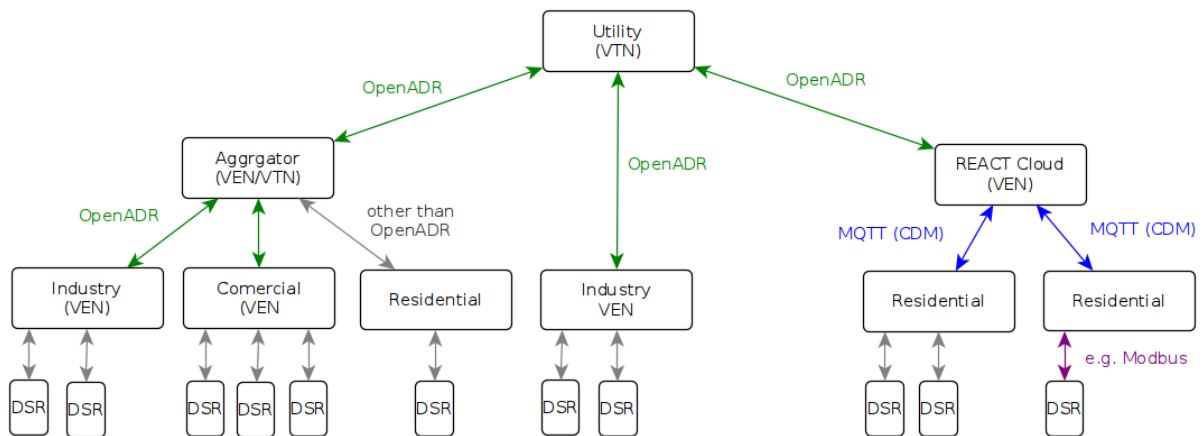


Figure 13: Overview of OpenADR actors and roles.

Likewise, OpenADR defines two different transport mechanisms for communication: HTTP and XMPP (XML Messaging and Presence Protocol). Nodes may communicate via HTTP either in push or pull mode. In push mode the VTN sends out information to the VEN, whereas in pull mode the VEN requests information from the VTN and initiates the communication. The actual information exchange is based on a set of different services defined by OpenADR. The most important services are the event service (EiEvent) and the report service (EiReport).

Event Service

When the VTN wants to trigger an event in push mode, it sends an `oadrDistributeEvent` to the VEN as shown in Figure 14. The major building blocks of this event are the `eventDescriptor`, `eiEventPeriod` and `eventSignals` as depicted in Figure 15.

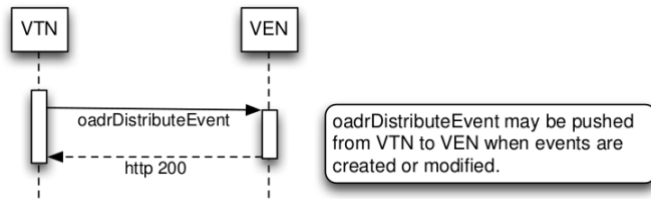


Figure 14: VTN sends events to VEN in push mode (OpenADR Alliance, 2015)

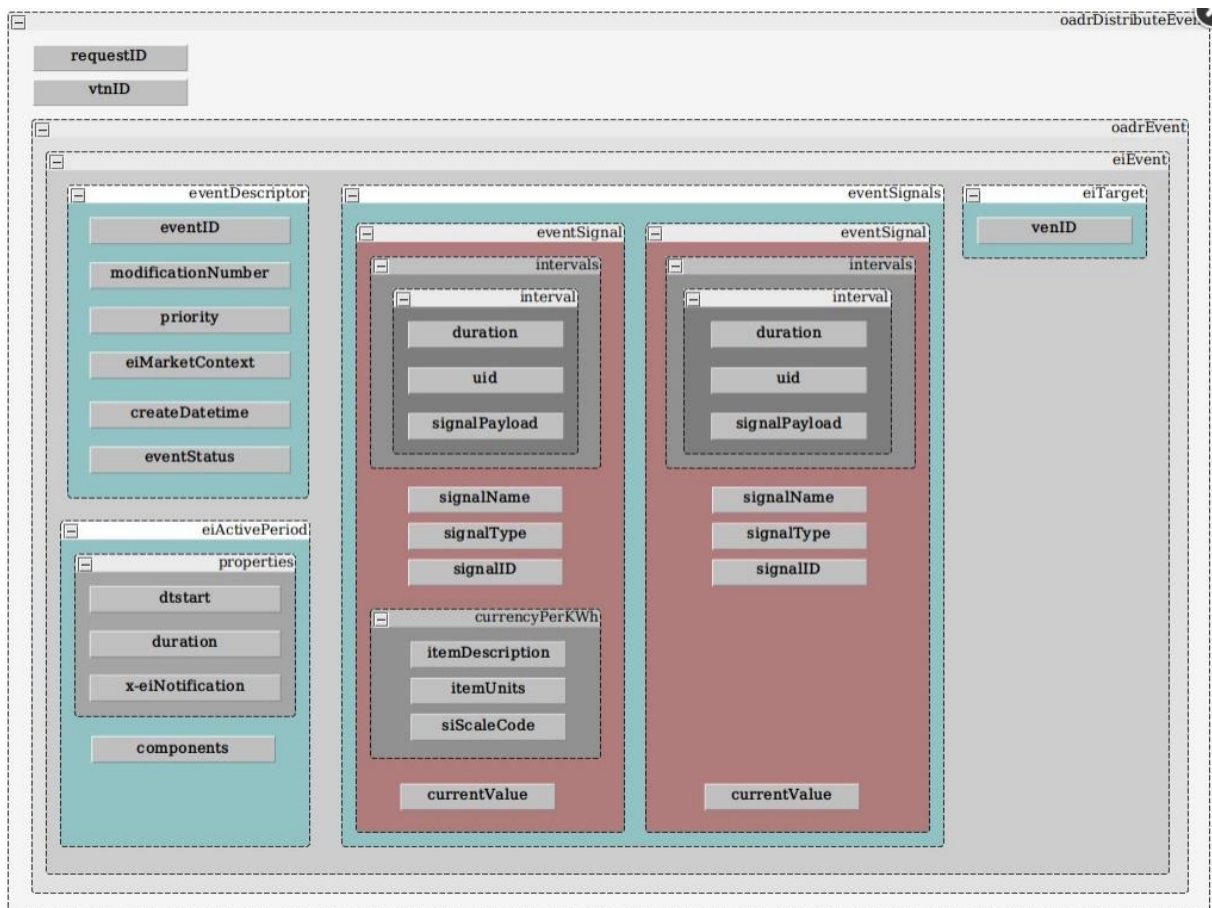


Figure 15: Building blocks of the general oadrDistributeEvent.

An event defines price or energy schedules for the active event period, which is indicated by its start and end time. For this period, several signals can be applied in the same event as a base price for the whole period or different price levels for individual time intervals within that period. Likewise, demand response actions, which are directly aiming to

energy and power changes, can be expressed by schedules. The following signal categories are defined in OpenADR:

- price electricity
- price of energy
- demand charge
- customer bid levels
- dispatch storage resources
- dispatch load
- control load

With regards to the REACT cloud platform, price signals from utilities could be integrated via OpenADR into the optimisation services to consider demand response actions according to the grid state. OpenADR supports different options to design the price signal. Price can refer to energy (currency/kWh) or power consumption (currency/kW) and can be expressed as absolute value, as delta or multiple of the existing price (Table 4).

Signal Category	Name (signal-Name)	Type (signalType)	units (item-Base)	Allowed Values	Description
Simple levels	SIMPLE	level	None	0,1,2,3	Simple levels
Price of electricity	ELECTRICITY_PRICE	price	currency/kWh	any	This is the cost of electricity expressed in absolute terms
	ELECTRICITY_PRICE	priceRelative	currency/kWh	any	This is a delta change to the existing price of electricity
	ELECTRICITY_PRICE	priceMultiplier	None	any	This is a multiplier to the existing cost of electricity
Price of energy	ENERGY_PRICE	price	currency/kWh	any	This is the cost of energy expressed in absolute terms
	ENERGY_PRICE	priceRelative	currency/kWh	any	This is a delta change to the existing price of energy
	ENERGY_PRICE	priceMultiplier	None	any	This is a multiplier to the existing cost of energy
Demand charge	DEMAND_CHARGE	price	currency/kW	any	This is the demand charge expressed in absolute terms
	DEMAND_CHARGE	priceRelative	currency/kW	any	This is a delta change to the existing demand charge
	DEMAND_CHARGE	priceMultiplier	None	any	This is a multiplier to the existing demand charge

Table 4: Pricing signals (OpenADR Alliance, 2015)

Another interesting signal regarding REACT could be the signal to dispatch storage resources (see Table 5). Depending on the demand response strategy and level of integration, OpenADR signals could be converted into the CDM format and directly communicated via MQTT to the storage resources without any optimisation at the REACT

cloud. On the other hand, OpenADR signals could also serve as target values, which in turn could be used as input parameter for optimisation services of the REACT cloud.

Signal Category	Name (signal-Name)	Type (signalType)	units (item-Base)	Allowed Values	Description
Used to dispatch storage resources	CHARGE_STATE	setpoint	energyXXX (1)	any	This is used to either charge or discharge a certain amount of energy from a storage resource until its charge state reaches a certain level.
	CHARGE_STATE	delta	energyXXX (1)	any	This is the delta amount of energy that should be contained in a storage resource from where it currently is.
	CHARGE_STATE	multiplier	None	0.0 < 1.0	This is the percentage of full charge that the storage resource should be at.

Table 5: Signal for storage resources. XXX represents real, apparent and reactive versions of power or energy (OpenADR Alliance, 2015)

Report Service

This service is used to report the so-called data points, which can be measured or calculated data. Usually, the VEN is the report producer, but reports from VTN to VEN are also possible. The reporting capabilities are exchanged during a report registration process, where a special METADATA report is created. This report holds all the different report types and data points that can be reported. OpenADR supports historical data reports and telemetry data reports. The later one is used for real time monitoring of data points but also for reporting the status of a resource.

REACT OpenADR interface

In Figure 16, we present the design of OpenADR interface in REACT which is used to receive the DR events from the relevant stakeholders. As it is shown, the VTN (e.g. aggregator, DSO, Utility, etc.) sends the oadrDistributeEvent message. Once the message is received by the VEN node, which can be implemented by using the open-source Python library openLEADR²⁸, the payload of the message is extracted and the data are saved in the database. The data represent the energy pricing or dispatch load information which can be then used by e.g. Energy optimization service to provide the optimal demand profiles. These profiles are then converted into control actions used to control the energy assets which are located in REACT pilot sites.

²⁸ <https://www.openleadr.org/>

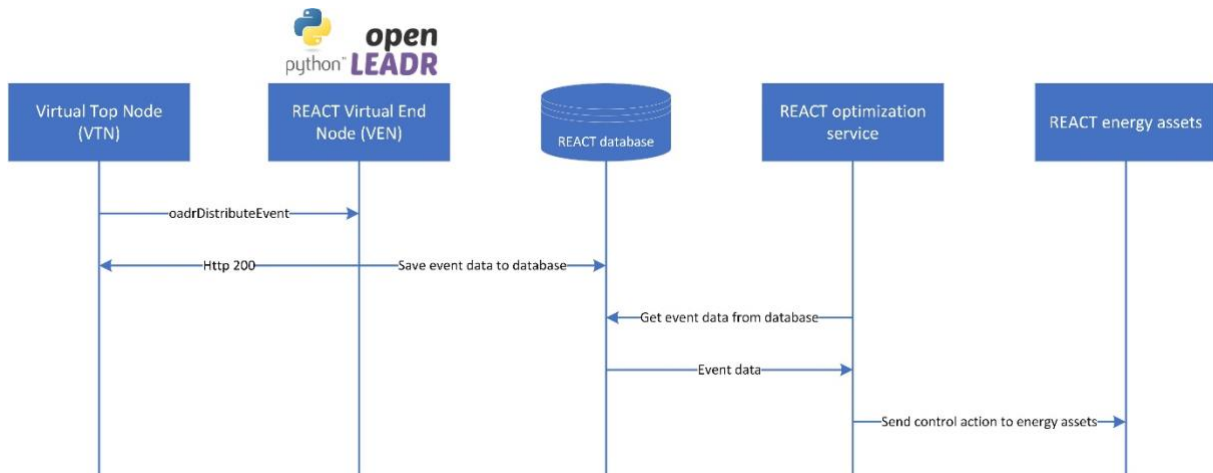


Figure 16 OpenADR oadrDistributeEvent

In Figure 17, we present the communication flow between VTN and VEN for the reports functionalities. This can be used e.g. so that Utility – VTN request the report on energy consumption from REACT, in order to check whether the requested DR event have been actually fulfilled. As can be seen, REACT energy assets continuously report different measurements (e.g. power, energy), which are stored in the database. REACT VEN provides the requested report, by first fetching the data from the database and providing them to VTN upon request.

A possible implementation of VEN on the REACT platform is provided in the code in Listing 4.

```

import asyncio
from datetime import timedelta
from openleadr import OpenADRClient, enable_default_logging

enable_default_logging()

async def collect_total_energy_aran():
    # This callback is called when there is a need to get the data from
    the database. Here the code would read the data from Influx database

    return get_aran_total_energy()

async def handle_event(event):
    # This callback receives an Event dict.
  
```

```
# The code below should save the data into database.
save_data_into_database(event)
return 'optIn'

# Create the client object
client = OpenADRClient(ven_name=react_node,

vtn_url='http://react.imp.bg.ac.rs:8080/OpenADR2/Simple/2.0b')

# Add the report capability to the client
client.add_report(callback=collect_total_energy_aran,
                  resource_id='pilot_aran',
                  measurement='energy',
                  sampling_rate=timedelta(seconds=10))

# Add event handling capability to the client
client.add_handler('on_event', handle_event)

# Run the client in the Python AsyncIO Event Loop
loop = asyncio.get_event_loop()
loop.create_task(client.run())
loop.run_forever()
```

Listing 4: VEN code implementation.

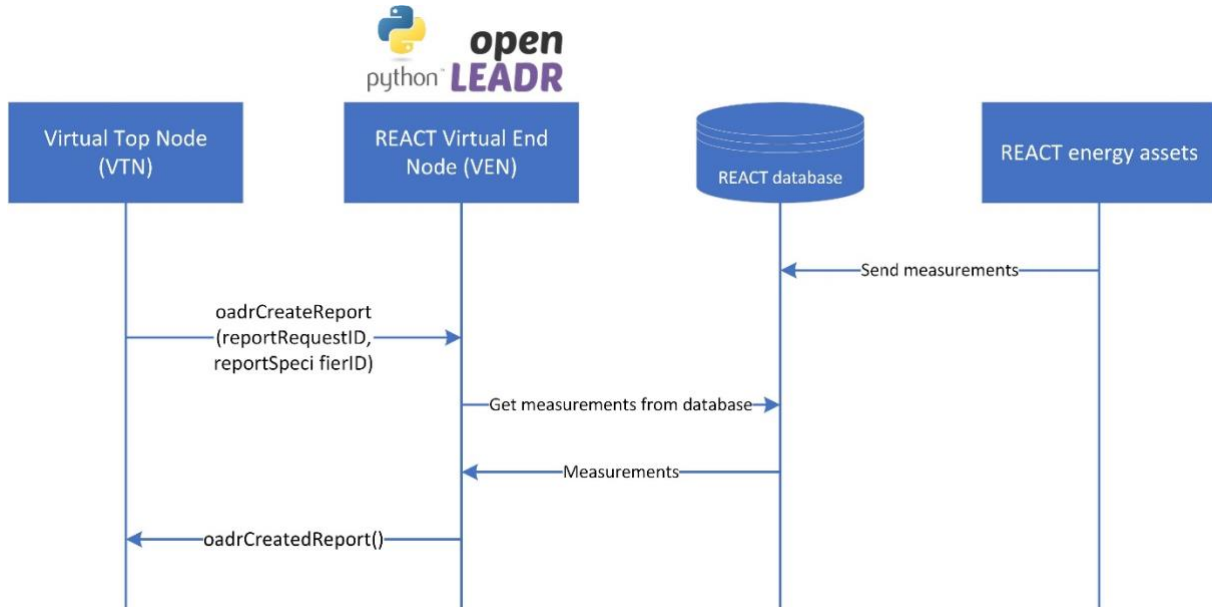


Figure 17 OpenADR oadrCreateReport

Summarizing, the aim of OpenADR is to connect the REACT cloud to grid side entities such as grid operators, service providers and utilities through a standardized communication protocol. This enables grid side entities to roll out demand response actions to the REACT resources like battery storage systems. For this, the REACT needs a new software module implementing the OpenADR VEN capabilities. Price signals serve as additional parameters for the optimisation services of the cloud. The reporting from REACT cloud to a VTN could be done quite straight forward since all historical and current measurements are stored in a database, as it has been described above.

4.2 Weather Service Integration

Different REACT services, such as energy production forecasts, require accurate weather forecasts for their correct functioning. After reviewing different web services that provide such functionality, weatherbit.io was selected since it provides all the necessary weather data with appropriate time resolution. In order to fetch the current weather observations and weather forecasts data from weatherbit.io service, a custom software component has been developed, as it is shown in Figure 18.

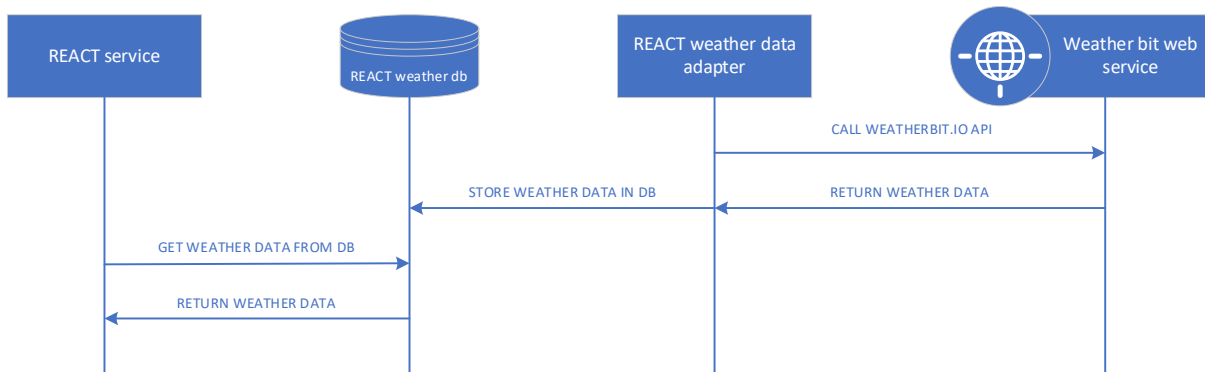


Figure 18 Weatherbit.io data adapter

As can be seen, REACT weather data adapter issues a call to weatherbit.io API, which then returns the data in a form of JSON object, with the fields described below. Next the data are processed in the adapter and stored in a relational MySQL database. Whenever a particular REACT service (e.g., production forecast service) requires weather forecast data, it performs a SQL query to the database.

Weatherbit.io web service offers different end points that provide the data such as: current weather, severe weather alerts, current air quality, historical weather, weather forecasts, etc. For the purpose of REACT project, we have selected the following end points:

- Current weather
- Daily weather forecast for 16 days
- Hourly weather forecast for 120 hours

The aforementioned REACT weather data adapter fetches data in a period manner:

- Current weather runs every hour
- Daily weather forecast runs every day
- Hourly weather forecast runs every day

The end points used to get the data are listed in Table 6

Daily weather forecast:	url: "https://api.weatherbit.io/v2.0/forecast/daily?key=".\$KEY."&units=M&lat=".\$LAT."&lon=".\$LON;
Hourly weather forecast:	url: "https://api.weatherbit.io/v2.0/forecast/hourly?key=".\$KEY."&units=M&lat=".\$LAT."&lon=".\$LON;

Current weather observation:	url: "http://api.weatherbit.io/v2.0/current?key=".\$KEY."&units=M&lat=".\$LAT." &lon=".\$LON;
------------------------------	---

Table 6 Weatherbit.io API endpoints (\$KEY - API key used for authorization, \$LAT and \$LON - latitude and longitude for the selected pilot location)

An example of response provided by weatherbit.io is given in Listing 5.

```
{
  "data": [
    {
      "wind_cdir": "NE",
      "rh": 59,
      "pod": "d",
      "lon": "-78.63861",
      "pres": 1006.6,
      "timezone": "America/New_York",
      "ob_time": "2017-08-28 16:45",
      "country_code": "US",
      "clouds": 75,
      "vis": 10,
      "wind_spd": 6.17,
      "wind_cdir_full": "northeast",
      "app_temp": 24.25,
      "state_code": "NC",
      "ts": 1503936000,
      "h_angle": 0,
      "dewpt": 15.65,
      "weather": {
        "icon": "c03d",
        "code": 803,
        "description": "Broken clouds"
      },
      "uv": 2,
      "aqi": 45,
      "station": "CMVN7",
      "wind_dir": 50,
    }
  ]
}
```

```

        "elev_angle":63,
        "datetime":"2017-08-28:17",
        "precip":0,
        "ghi":444.4,
        "dni":500,
        "dhi":120,
        "solar_rad":350,
        "city_name":"Raleigh",
        "sunrise":"10:44",
        "sunset":"23:47",
        "temp":24.19,
        "lat":"35.7721",
        "slp":1022.2
    }
],
"minutely":[ ... ],
"count":1
}

```

Listing 5: Weatherbit payload response.

The meaning of the fields in the JSON object above is given as follows:

- count: Count of returned observations.
- data:
 - lat: Latitude (Degrees).
 - lon: Longitude (Degrees).
 - sunrise: Sunrise time (HH:MM).
 - sunset: Sunset time (HH:MM).
 - timezone: Local IANA Timezone.
 - station: Source station ID.
 - ob_time: Last observation time (YYYY-MM-DD HH:MM).
 - datetime: Current cycle hour (YYYY-MM-DD:HH).
 - ts: Last observation time (Unix timestamp).
 - city_name: City name.
 - country_code: Country abbreviation.
 - state_code: State abbreviation/code.

- pres: Pressure (mb).
- slp: Sea level pressure (mb).
- wind_spd: Wind speed (Default m/s).
- wind_dir: Wind direction (degrees).
- wind_cdir: Abbreviated wind direction.
- wind_cdir_full: Verbal wind direction.
- temp: Temperature (default Celsius).
- app_temp: Apparent/"Feels Like" temperature (default Celsius).
- rh: Relative humidity (%).
- dewpt: Dew point (default Celsius).
- clouds: Cloud coverage (%).
- pod: Part of the day (d = day / n = night).
- weather: {
 - icon: Weather icon code.
 - code: Weather code.
 - description: Text weather description.
- vis: Visibility (default KM).
- precip: Liquid equivalent precipitation rate (default mm/hr).
- snow: Snowfall (default mm/hr).
- uv: UV Index (0-11+).
- aqi: Air Quality Index [US - EPA standard 0 - +500]
- dhi: Diffuse horizontal solar irradiance (W/m²) [Clear Sky]
- dni: Direct normal solar irradiance (W/m²) [Clear Sky]
- ghi: Global horizontal solar irradiance (W/m²) [Clear Sky]
- solar_rad: Estimated Solar Radiation (W/m²).
- elev_angle: Solar elevation angle (degrees).
- h_angle: Solar hour angle (degrees).

4.3 Open API Concept

The Open Application Programming Interface (Open API) is commonly defined as an API that uses a common or universal language or structure to promote more universal access.

Within REACT, the Open API is used to support REACT platform connectivity, enabling exploitation of data provided by smart grid and external (third party) services (such as provision of energy prices, weather data, data exchanging with DR aggregators, etc.). In addition, the Open API will allow for reuse of the REACT analytical services by putting them

at disposal to third parties. This will allow an easy uptake and replication of leveraging concepts and extension of third-party business processes and their further integration into the smart grid products and services.

Aiming to maximize interoperability and ease of integration, specification of REACT analytical services follows OpenAPI, a specification and initiative for the creation of human and machine-readable interface files, used to describe, produce, consume, and visualize RESTful web services. Being both human and machine readable allows both people and computers to discover and understand the capabilities of a service without requiring access to source code, additional documentation, or inspection of network traffic. As described in deliverable D6.2, OpenWhisk, which is used for the deployment of analytical services, provides a generic REST API in OpenAPI format.

The OpenAPI Specification (OAS) specifies the rules and syntax required to describe the API's interface. It continues to introduce updates to make the specification simpler to use, and easier for humans and computers to understand. The general outline of an OAS defined API using the latest, OAS 3 is shown in the next figure.

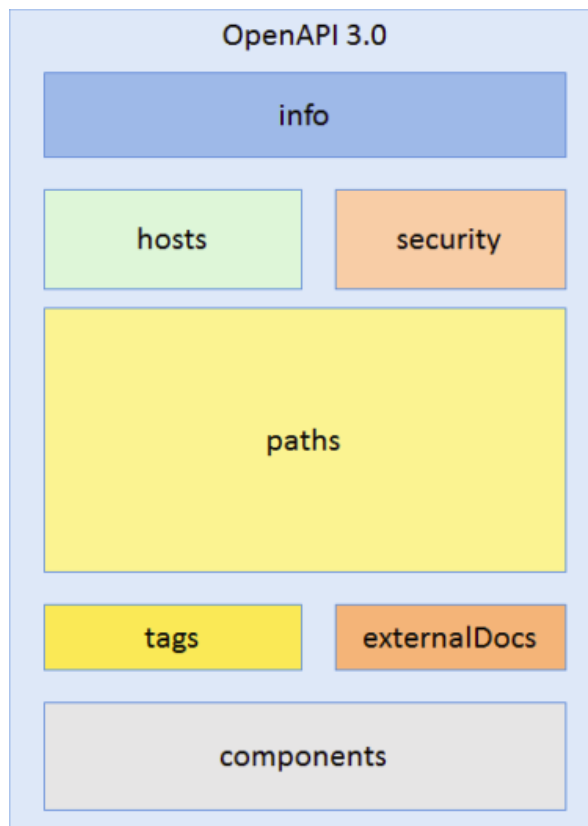


Figure 19 OAS general outline

The above figure breaks down the various sections in an API contract designed by the OAS. The meaning of each section is as follows:

- Info - Contains the meta data associated with the API's contract. The required parts of this section are the title, version, and description of the API. This section can also have other fields like contact information, license information and terms of service links.
- Servers - The API is the contract between the consumer and the server. The Server object can give client information on where the API's servers are located, through its URL. OAS 3.0 supports definition of multiple servers, which is useful since in the real world, APIs exist in multiple environments, and the contract's business logic may change depending on the environment.
- Security - Every API needs some level of security. The OpenAPI description format supports various authentication and authorization schemes to mitigate unknown, unregistered users from accessing the API. The OpenAPI supports:
 - HTTP authentication schemes
 - API keys in headers, cookies, or query strings
 - OAuth2
 - OpenID
- Paths - This section shows the various end points that the API exposes, and the corresponding HTTP methods. It's also under each method that the actual request-response cycle is detailed. The requests are described by Parameters objects, and the responses by the Responses objects.
 - Parameters - the variable parts of the request. There are four types of parameters that can be specified: path parameters (such as /users/{id}), query parameters (such as /users?role=admin), header parameters (such as X-MyHeader: Value), cookie parameters (such as Cookie: debug=0; csrftoken=BUSe35dohU3O1MZvDDU).
 - Responses - the objects returned on a request. Every response is defined by its HTTP status code (defining whether the request was successful or unsuccessful) and the data returned.
- Components - The API may repeat a lot of existing parameters or response descriptions in many different paths and operations and rewriting them every time makes them prone to inconsistent descriptions and can be very time consuming. The component object can hold a set of reusable objects of the API's design. The reusable objects can be schemas, responses, parameters, examples and more.
- External Docs - Any additional information that can be offered to ease consumption and integration with the API.
- Tags - Friendly categories to group various operations. This allows consumers of the API to better segment and identify what they want to use the API for. These

tags can also be handled by other third-party tools which integrate or read the OAS.

The API for REACT analytical services will be specified in detail after deployment of the services on the cloud platform. This will be accomplished as part of the collaboration with WP5 activities.

5. Conclusions

This deliverable document summarizes the work done in the definition of the Semantic data model and the connectivity with smart-grid services of the REACT project. This Semantic data model, in the form of an ontology, ensures the formal representation with unambiguous meaning of the data gathered in the different pilot sites, as well as their relationship with the REACT platform core services and tools.

The ontology is developed following an ontology engineering methodology and considering the Semantic Web best practices. The REACT ontology is based on ontology design patterns, reuses well-known ontologies, and follows a modular approach. Additionally, the careful documentation provided, and the complete metadata related to the terms and concepts defined in the ontology contributes to the ontology's understanding. Last but not least, it is publicly available to foster its reusability by other research projects and users dealing with similar problems faced in the REACT project.

The representation of the three pilot sites with ontology terms and its storage in a centralized repository allows an easy integration with the core services and tools and provides a common view of all heterogeneous sets of data and equipment installed in the pilots.

This deliverable also defines the Open API concept to be applied for the integration with smart-grid services, and the integration with external services to gather required data for the successful implementation of the REACT platform core services.

References

- Allemang, D., & Hendler, J. (2011). *Semantic web for the working ontologist: effective modeling in RDFS and OWL*. Elsevier. doi:10.1016/C2010-0-68657-3
- Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific american*, 284(5), 34-43.
- Borgo, S., & Masolo, C. (2009). Foundational choices in DOLCE. In S. Borgo, & C. Masolo, *Handbook on ontologies* (pp. 361-381). Springer.
- Breitman, K. K., Casanova, M. A., & Truszkowski, W. (n.d.). Ontology in Computer Science. In K. K. Breitman, M. A. Casanova, & W. Truszkowski, *Semantic Web: Concepts, Technologies and Applications* (pp. 17-34). Springer London.
- d'Aquin, M., Schlicht, A., Stuckenschmidt, H., & Sabou, M. (2009). Criteria and Evaluation for Ontology Modularization Techniques. In H. Stuckenschmidt, C. Parent, & S. Spaccapietra, *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization* (pp. 67-89). Springer Berlin Heidelberg.
- Domingue, J., Fensel, D., & Hendler, J. A. (2011). Introduction to the Semantic Web Technologies. In J. D. Hendler, *Handbook of Semantic Web Technologies* (pp. 1-41). Berlin: Springer Berlin Heidelberg.
- Ensan, F., & Du, W. (2013). A Semantic Metrics Suite for Evaluating Modular Ontologies. *Inf. Syst.*
- Eснаоla, I., Fernandez, I. a., Ferreiro, S., Gomez, M., Lázaro, I., & García, Á. (2019). Towards Animal Welfare in Poultry Farms through Semantic Technologies. *IoT Connected World & Semantic Interoperability Workshop (IoT-CWSI) 2019*. Spain.
- Eснаоla-Gonzalez, I., Bermúdez, J., Fernandez, I., & Arnaiz, A. (2020). Ontologies for Observations and Actuators in Buildings: A Survey. *Semantic Web*.
- Eснаоla-Gonzalez, I., Bermúdez, J., Fernandez, I., & Arnaiz, A. (2021). EEPsA as a core ontology for energy efficiency and thermal comfort in buildings. *Applied Ontology*, 16(2), 193-228.
- Fernández-López, M., Suárez-Figueroa, M. C., & Gómez-Pérez, A. (2012). Ontology Development by Reuse. In M. C.-F.-P. Gangemi, *Ontology Engineering in a Networked World* (pp. 147-170). Springer Berlin Heidelberg.
- Garijo, D. (2017). WIDOCO: A Wizard for Documenting Ontologies. *The Semantic Web -- ISWC 2017*, (pp. 94-102).

- Grau, B. C., Horrocks, I., Kazakov, Y., & Sattler, U. (2008). Modular reuse of ontologies: Theory and practice. *Journal of Artificial Intelligence Research*, 273-318.
- Guarino, N. (1998). *Formal Ontology and Information Systems*. IOS Press.
- Gyrard, A., Zimmermann, A., & Sheth, A. (2018). Building IoT based applications for Smart Cities: How can ontology catalogs help? *IEEE Internet of Things Journal*.
- Heath, T., & Bizer, C. (2011). *Linked data: Evolving the web into a global data space*. Morgan & Claypool Publishers.
- Hitzler, P., Gangemi, A., & Janowicz, K. (2016). *Ontology Engineering with Ontology Design Patterns: Foundations and Applications*. IOS Press.
- Lassila, O., & McGuinness, D. (2001). The role of frame-based representation on the semantic web. *Linköping Electronic Articles in Computer and Information Science*.
- Lozano-Tello, A., & Gómez-Pérez, A. (2004). Ontometric: A method to choose the appropriate ontology. *Journal of Database Management (JDM)*, 1-18.
- OpenADR Alliance. (2015). *OpenADR 2.0 Profile Specification B Profile*.
- Peroni, S., Shotton, D., & Vitali, F. (2013). Tools for the Automatic Generation of Ontology Documentation: A Task-Based Evaluation. *Int. J. Semant. Web Inf. Syst.*
- Pinto, H. S., Staab, S., & Tempich, C. (n.d.). DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolving Engineering of ontologies. *16th European Conference on Artificial Intelligence (ECAI)* (pp. 393-397). IOS Press.
- REACT D6.1, I. L. (n.d.). Retrieved from <https://react2020.eu/download/d6-1-react-system-reference-architecture/>
- Simperl, E. (2009). Reusing ontologies on the Semantic Web: A feasibility study. *Data & Knowledge Engineering*, 905-925.
- Stuckenschmidt, H., & Klein, M. (2007). Reasoning and change management in modular ontologies. *Data & Knowledge Engineering*, 200 - 223.
- Studer, R., Benjamins, V. R., & Fensel, D. (1998). Knowledge engineering: principles and methods. *Data and knowledge engineering*, 161-198.
- Suárez-Figueroa, M. C., Gómez-Pérez, A., Motta, E., & Gangemi, A. (2012). The NeOn Methodology for Ontology Engineering. In *Ontology Engineering in a Networked World* (pp. 9-34). Springer Berlin Heidelberg.

Appendix I – REACT ORSD

This Appendix shows an excerpt of the REACT ontology requirements specification document (ORSD).

Ontology Requirements Specification Document	
1	Purpose
	Represent all the necessary knowledge to support the achievement of island energy independence through renewable energy generation and storage, a demand response platform
2	Scope
	It must cover buildings used for different purposes (residential, commercial, ...) as well as equipment of different types that may be involved in energy generation and storage such as batteries or PV panels.
3	Implementation Language (optional)
	OWL
4	Intended End-Users (optional)
	<ul style="list-style-type: none"> • REACT platform users • Decision-makers
5	Intended Uses
	<ul style="list-style-type: none"> • Represent relevant data in a homogeneous and unambiguous manner • Provide information to the Optimization service • Provide information to the REACT Visualization tools • (...)
6	Ontology Requirements

	a. Non-Functional Requirements
	The Ontology must be written following the CamelCase naming convention.
	b. Functional Requirements: Lists or tables of requirements written as Competency Questions and sentences
	<ul style="list-style-type: none"> • Which are the qualities observed by a given equipment? • Which is the gateway of a given installation? • When did a given installation join the REACT platform? • When was a given equipment installed? • (...)
7	Pre-Glossary of Terms (optional)
	a. Terms from Competency Questions
	Qualities Equipment Gateway (...)
	b. Terms from Answers
	InstallationDate Execution Space (...)

Appendix II – REACT ontology in use example

RDF representation of the REACT ontology instantiation for a given house participating in the REACT project.

```

@prefix aff: <https://w3id.org/affectedBy#> .
@prefix bot: <https://w3id.org/bot#> .
@prefix dbo: <https://dbpedia.org/ontology/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix dev: <https://react2020.eu/device/> .
@prefix eep: <https://w3id.org/eep#> .
@prefix foi4eepsa: <https://w3id.org/eepsa/foi4eepsa#> .
@prefix inst: <https://react2020.eu/LaGraciosa/> .
@prefix qudt: <http://qudt.org/schema/qudt/> .
@prefix react: <https://w3id.org/react#> .
@prefix respond: <https://w3id.org/respond#> .
@prefix saref: <https://w3id.org/saref#> .
@prefix unit: <http://qudt.org/vocab/unit/> .

inst:LG27 a
    bot:Space , aff:FeatureOfInterest ;
    dc:identifier "LG27" ;
    dbo:membershipAsOf "Wed Dec 02 00:00:00 CET 2020" ;
    bot:hasElement dev:LG27_VIC-GXHQ2022T1Y6A-100 ,
dev:LG27_VIC-GXHQ2022T1Y6A-225 , dev:LG27_VIC-GXHQ2022T1Y6A-227 ,
dev:LG27_VIC-GXHQ2022T1Y6A-223 , dev:LG27_VIC-GXHQ2022T1Y6A-30 ,
dev:LG27_VIC-100000006c8efa84 ;
    foi4eepsa:hasMarketCategory "residential" ;
    react:hasDemandEnergyCarrier "electric" ;
    react:hasGenerationEnergyCarrier "electric" ;
    react:hasMaxGridExport "" ;
    react:hasMaxGridImport "" ;
    react:hasTopology "DC Coupled" .

dev:VIC-GXHQ2022T1Y6A-100
    a
    eep:Executor ;
    dc:identifier "GXHQ2022T1Y6A-100" ;
    react:hasEquipmentCategory "ControlUnit" ;
    react:hasInstallationDate "Wed Dec 02 00:00:00 CET 2020" ;
    react:hasSerialNumber "" ;
    respond:connectsToInternetThrough
        dev:LG27_VIC-100000006c8efa84 ;
    respond:hasModel "Cerbo GX" ;
    saref:hasManufacturer "Victron Energy" .

inst:LG27_pAcHome_Execution

```

```
a eep:Execution ;
qudt:unit unit:W ;
eep:madeBy dev:LG27_VIC-GXHQ2022T1Y6A-100 ;
eep:onQuality inst:LG27_pAcHome ;
react:hasAccessType "Read" ;
react:hasMeasurementIndex "1" ;
react:hasSamplingMethod "N/A" .

inst:LG27_pAcHome a aff:Quality ;
dc:identifier "pAcHome" ;
aff:belongsTo inst:LG27 .
```