

H2020 – LC – SC3 – EE – 2019 – GA 894240

Operating System for Smart Services in Buildings



## D2.3 Functional Specification of the IoT-Ecosystem

WP2 IoT for Smart Buildings

	Name	Date
Prepared by	Brian Nielsen (AAU), Søren Enevoldsen (AAU), Junior Dongo (AAU), Amir Laadhar (AAU), Nicola Cibir (AAU)	14.01.2022
Peer reviewed by	Frédéric Revaz (HES-SO)	21.02.2022
Reviewed and approved by	Dominique Gabioud (HES-SO)	24.02.2022



## Distribution list

External		Internal	
European Commission	x	Consortium partners	x

## Change log

Version	Date	Remark / changes
1.1	14.01.2022	First Version by AAU
2.1	07.02.2022	Second version by AAU
3.1	24.02.2022	Final version by HES-SO

## To be cited as

“D2.3 Functional Specification of the IoT-Ecosystem” of the HORIZON 2020 project domOS, EC Grant Agreement No 894240.

## Disclaimer

This document's contents are not intended to replace the consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided “as is” and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

## Table of contents

<b>1.</b>	<b>Introduction .....</b>	<b>6</b>
1.1.	Background .....	6
1.1.1.	New Smart Services .....	6
1.1.2.	Digitalization of Buildings .....	6
1.1.3.	Interoperability .....	6
1.1.4.	Privacy and Reliability .....	7
1.1.5.	domOS Ecosystem and IoT platforms .....	7
1.2.	Objective .....	8
1.3.	The Process .....	9
1.4.	Definitions .....	10
<b>2.</b>	<b>Overall Architecture and domOS Reference Model .....</b>	<b>12</b>
2.1.	Layered Model .....	12
2.2.	Main Functional Principles .....	13
2.2.1.	WoT Compliance .....	13
2.2.2.	Building Description .....	13
2.2.3.	Device (Things) Management .....	14
2.2.4.	Application Management .....	14
2.2.5.	Security and Privacy .....	15
2.2.6.	User- and Administrative Interfaces .....	15
2.3.	domOS Eco-System (Normative) .....	16
<b>3.</b>	<b>Platform Instantiation Guidelines (Non-normative) .....</b>	<b>18</b>
3.1.	Use-Case Diagram .....	18
3.1.1.	Use-Cases .....	19
3.2.	System Structure .....	20
3.2.1.	Components .....	20
3.2.2.	Main Interfaces .....	25
3.3.	Behaviour .....	26
3.3.1.	Adding a WoT Device .....	26
3.3.2.	Normal Scenario Where an Application Accesses (reads) a WoT Property .....	27
3.3.3.	An Event Occurs in a WoT Device .....	28
3.4.	IoT Semantic Interoperability Using Semantic Web Technologies .....	29
<b>4.</b>	<b>Key Interface Specifications (Normative) .....</b>	<b>32</b>
4.1.1.	Swagger Key Interfaces Specifications .....	32
4.1.2.	TDD API .....	33
4.1.3.	Building Description Directory .....	33
4.1.4.	Intermediary API .....	34
4.1.5.	Authentication .....	34
4.2.	Building Description .....	34
<b>5.</b>	<b>Conclusion and Future Work .....</b>	<b>36</b>
5.1.	Conclusion .....	36



5.2.	Future work .....	36
<b>6.</b>	<b>References.....</b>	<b>37</b>
	<b>Appendix A: Requirements Review and Consolidation .....</b>	<b>38</b>
	<b>Appendix B: Requirements Links and Traceability .....</b>	<b>45</b>
<b>7.</b>	<b>Objectives and Success Criteria.....</b>	<b>46</b>
7.1.	Overview.....	47
7.1.1.	Overview of the IoT Ecosystem .....	47
7.2.	Functional Requirements.....	48
7.2.1.	Requirements for Platforms .....	48
7.2.2.	Requirement for Applications.....	48
7.2.3.	Requirement for Smart Systems.....	48
7.2.4.	Security Requirements .....	49
7.2.5.	Requirements on Semantics.....	49
7.2.6.	Requirements on Privacy.....	49
7.2.7.	Requirements on Deployment Topology.....	50
7.2.8.	Support of Privacy.....	51
7.3.	Non-Functional Requirements.....	51
7.3.1.	Compliance with Recognized IoT / Web Standards.....	51
7.3.2.	Safety .....	52
7.3.3.	Usability .....	53
7.3.4.	Performance .....	53
7.4.	Smart System Description.....	53
7.5.	Subscription to an Application.....	53
7.6.	Intermediary Function .....	54
7.7.	Smart Systems Directory .....	55

## List of figures

Figure 1: SGAM Interoperability Layers (Reproduced from (CEN-CENELEC-ETSI Smart Grid Coordination Group, 2012)) .....	7
Figure 2: domOS Eco-System Development Phases .....	9
Figure 3: Layered Model of the domOS IoT Eco-System.....	12
Figure 4: High-Level Use-Cases for the domOS Platform.....	18
Figure 5: domOS Component Diagram .....	21
Figure 6: Components involved in WoT Interactions.....	26
Figure 7: Adding a new Thing Description .....	26
Figure 8: Application Accessing a WoT Property Affordance in an open platform.....	27
Figure 9: Application Accessing a WoT Property in a Closed Platform .....	28
Figure 10: Application Listening to Events.....	29



Figure 11: Semantic Search Sequence Diagram.....	30
Figure 12: Compliance Check Sequence Diagram .....	31
Figure 13: Example Swagger Interface.....	32
Figure 14: Principal Structure of the Building Description.....	34
Figure 15: Example of a Building Description in JSON-LD .....	35
Figure 16: Elements in the IoT Ecosystem .....	47
Figure 17: Two Possible Topologies: (a) Distributed Cloud Solution (B) Edge Solution.....	50
Figure 18: Illustration of the Intermediary concept in the WoT architecture (W3C (WoT Architecture), 2020) .....	54
Figure 19: Implementation of the intermediary based on the servient concept (W3C (WoT Architecture), 2020) .....	54

## List of tables

Table 1: Definitions .....	10
Table 2: domOS Eco-System Specification .....	16
Table 3: Requirements Review.....	39
Table 4: Requirements Traceability Labels .....	45
Table 5: Success Criteria.....	46
Table 6: Status of W3C WoT standards.....	51

## Terms, definitions, and abbreviated terms

DMP	Data Management Plan
DPIA	Data Protection Impact Assessment
DPO	Data Protection Officer
GDPR	General Data Protection Regulation
PC	Project Coordinator
TRL	Technology Readiness Level
RDF	Resource Description Framework
WoT	Web of Things
DOPL	domOS Platform
dCO	domOS Common Ontology

# 1. Introduction

## 1.1. Background

### 1.1.1. New Smart Services

Soon new automated intelligent digital services will help to improve the energy efficiency, energy flexibility, comfort, and convenience of residential buildings needed to achieve the urgent sustainability targets. They will help integrating buildings with the current energy systems as well as enable new models based on integrated energy systems, large scale renewal sources, and local energy communities. These services typically optimize by combining several knowledge sources such as weather forecast, production- and consumption-forecasts (at local, regional, or national levels). They will typically be deployed as remote or cloud services that require access via the Internet to sensing, automation, and control offered by the buildings.

### 1.1.2. Digitalization of Buildings

Whilst digitizing new and modern high-end buildings is common, retrofitting existing (of which there are most) buildings and appliances is more challenging. One challenge is that the building installations are more cumbersome to instrument and more difficult to establish interference free efficient communication. A related challenge is that the upgrade technologies are often siloed and non-interoperable that leads to islanded operation. This contradicts the needs of the envisioned future smart services. We see the Internet of Things (IoT) as a key enabler for smart services.

### 1.1.3. Interoperability

Providing automated access to devices from multiple different vendors requires that the devices are open and interoperable a priori or that they can be made interoperable by providing a suitable abstraction or conversion layer(s).

Interoperability is the ability of two or more devices from the same vendor, or different vendors, to exchange information and use that information for correct co-operation (IEC61850-2010).

It is important to recognize that interoperation goes beyond basic network connectivity and protocols, but extends to semantic based information exchange and functionality, as captured by the so-called interoperability layers. As example we provide Figure 1 that presents the key interoperability layers for the SGAM Smart Grid Architecture Model. domOS focus mostly on the communication layer (via the Web of Things standard) and information layer (via the domOS common ontology, see (domOS D3.2, 2021)).

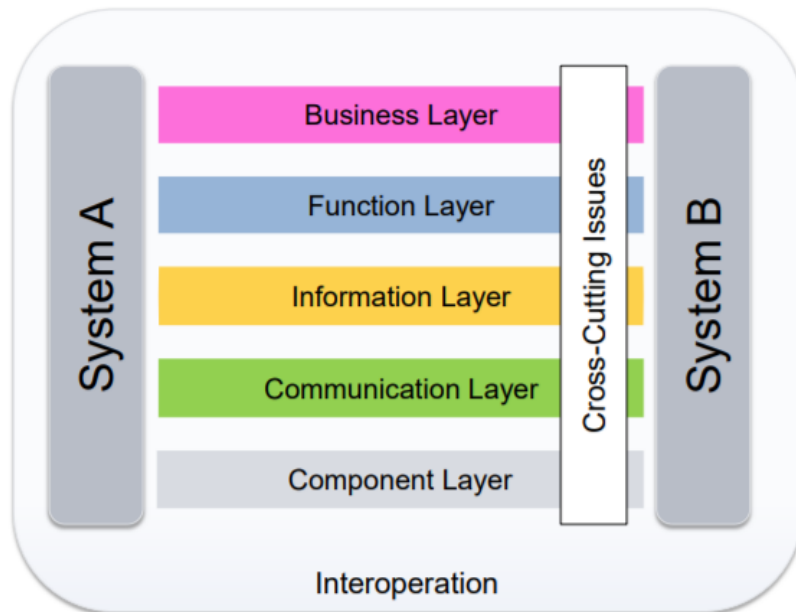


FIGURE 1: SGAM INTEROPERABILITY LAYERS (REPRODUCED FROM (CEN-CENELEC-ETSI SMART GRID COORDINATION GROUP, 2012))

#### 1.1.4. Privacy and Reliability

Opening – in particular private – buildings for remote or third-party sensing and actuating, requires that the access must be strictly controlled for privacy and security reasons. The data collected from these sensors, even as metering data, may potentially be abused to infer or leak information about occupancy behaviour. Only authenticated applications should be allowed to receive building data, and only the minimum data needed for their operation.

Reliability is a further challenge: devices may fail and need replacement, batteries are not replaced, software may be buggy, networks and Internet access fails intermittently, remote clouds and services also fail. In any non-trivial distributed system, such failures should be considered normal events, and not exceptional. The whole architecture and model for service delivery must be constructed to tolerate such failures, and fall-back to local control loops, that preserve a certain level of functionality, albeit less optimised.

#### 1.1.5. domOS Ecosystem and IoT platforms

From a domOS perspective an **IoT-eco-system** is an open collection of collaborating IoT-devices, IT-services, applications, and system components used to deliver smart (energy, living, etc.) services to buildings.

An **IoT platform** is the required hardware and system software components needed to facilitate the delivery a business service to a set of costumers. The platform is thus a central part of an eco-system that bridges applications and IoT devices. By developing the platforms from common concepts and specifications, the scope of IoT devices and applications are greatly improved and makes porting between platform instances easier.

A **platform instance** is a specific hardware and software implementation of the common concepts developed and delivered by a given vendor or a given platform operator.

Platform instances can vary greatly in application characteristics and range in deployment size, resulting in different instances. As illustrative examples, the platform instances can range in complexity from simple to very complex:

**Simple Gateway:** A single smart-service is delivered through a home gateway, executing a single application, e.g., a heating optimization service. It is typically supplied from the service provider when a subscription is engaged and governed by a manually engaged contract.

**Closed Platform:** A moderate number of applications are delivered under the responsibility of a single service provider typically to a larger collecting of buildings or apartments. Hence, being a **closed platform** (attached applications and devices are controlled by, and under responsibility of, the service provider also acting as platform operator) makes it easier for (but also the responsibility of) the vendor to ensure privacy and security on behalf of its customers. But the platform must scale sufficiently to handle many buildings and customers performance- and management-wise. The platform is typically deployed in the service provider's cloud. Applications are activated by signing a contract with the service provider.

**Open Platform:** The building owner is in control of the dynamic set of devices attached to his building, and the dynamic set of applications he subscribes to. Applications may be developed by 3<sup>rd</sup> party developers and potentially be advertised in an application marketplace and activated by the building owner on demand. In such a platform strong security, access control, and privacy management features must be implemented. The platform is deployed in a cloud and operated by an independent platform operator.

The domOS project will prototype three different platform instances (SION, Cloud.IO, Arrowhead) having different application areas, deployment scale, and demonstrator goals. The functional specification must consequently be tailorable and allow sufficient flexibility to cater for the needs of the different platforms whilst providing sufficient similarity to facilitate industrialization of smart buildings and services.

## 1.2. Objective

To meet these overall challenges, the project will define the domOS IoT eco-system based on the high-level requirements defined in "D2.1 Report on Requirement Analysis for IoT Ecosystems" (domOS D2.1, 2021). This entails:

- A reference-model that shows the conceptual layering of the eco-system.
- A classification of what features are mandated and optional
- Definition of the major components and roles.
- Suggested implementation structure, behaviour, and technologies.
- Concise specification of the core concepts, interfaces, and APIs.



### 1.3. The Process

The development of the eco-system progress according to the overall plan illustrated in Figure 2. Prior to the functional specification, the technology foundation has been reviewed, a set of high-level requirements formulated, the first ontology version proposed (see (domOS D3.2, 2021)), and proof-of-concept of WoT intermediaries and things directory prototypes have been implemented to serve as additional input.

As high-level requirements are typically neither directly implementable, consistent, nor complete, they have been reviewed and consolidated. To identify the core requirements for further functional specification of the eco-system, the MOSCOW prioritization principles was applied. The details of this particular process are exposed in Appendix A and B.

Based on these specifications, the definition of each platform instance follows a similar process. As each platform is tailored for different contexts and targets different objectives, each has its *own* concept definition, requirements, and functional specifications. It is assumed that each instance is open to a priori unknown applications and a priori unknown smart systems, but the extend is defined in its own concept definition.

In its requirements specification, each Platform decides to rely on the domOS ecosystem to implement the openness, and in its functional specification, each Platform takes over the Core features of the domOS ecosystem specification and implements by its own means the Instance features.

The process will continue by detailed technical design and development of the three domOS platform instances that will be evaluated and demonstrated by the use-cases on flexible energy services.

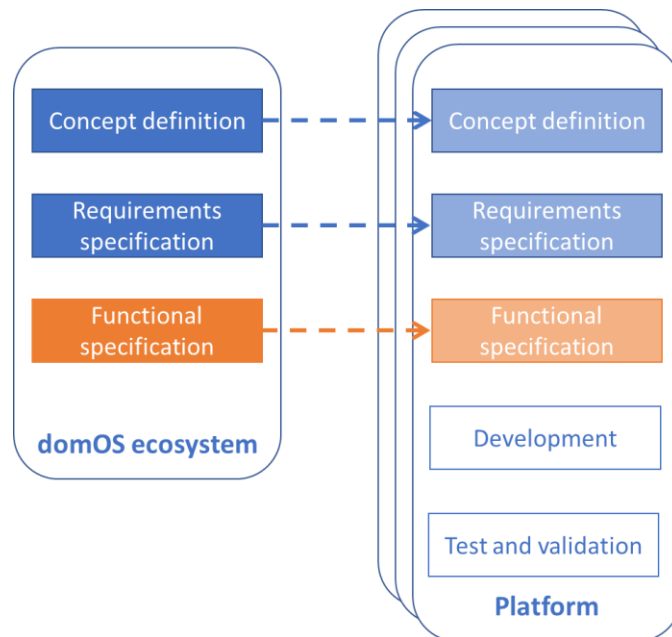


FIGURE 2: DOMOS ECO-SYSTEM DEVELOPMENT PHASES

## 1.4. Definitions

TABLE 1: DEFINITIONS

Connected system	Digital system in a <b>Container</b> providing a data interface for an external information system	A Wi-Fi heat pump, a SunSpec compatible solar inverter, a Zaptec charging station for electrical vehicle, a Z-Wave controller with its wireless peripherals
Smart system	Connected system made compliant with the domOS IoT ecosystem	A connected system together with a cloud hosted domOS adapter component.
domOS WoT thing	domOS compliant WoT thing	Thermostat as defined in the domOS ontology (dco:thermostat)
Customer	Natural (or legal) person in charge of the management of a <b>Container</b> and using a <b>Service</b> .	An adult inhabitant in a household, a facility operation in a service building or in a multi-family residential building
Container	In this context, premises where the collection of <b>Smart systems</b> managed by a <b>Customer</b> are located: an apartment in a multi-family residential building, a single-family house, the communal area in a building...	The single-family house where the four above mentioned smart systems are located.
Application	Software component interacting with one or more <b>Smart systems</b> located in a <b>Building</b> .	A component aiming at maximising the self-consumption of solar power by the heat pump and the electrical vehicle charging station
Service provider	Service Provider delivers a business service through an application	NeoGrid
Business-Service	Interplay of an <b>Application</b> and of one or several <b>Smart systems</b> , generating an added value for a <b>Customer</b> .	Self-consumption optimisation
SoA-Service	A software component that can function independently and offer a certain functionality via an API to other software client components	Energy consumption forecast service Energy price forecast service Public building information registry (e.g., Danish "BBR")
Platform	Software environment acting as a mediation entity between on one side <b>Containers</b> and their <b>Smart systems</b> and on the side <b>Applications</b> .	A cloud hosted utility enabling self-consumption optimisation application to interact with the building hosting a solar inverter, a heat pump and an electrical vehicle charging station.
domOS IoT ecosystem	ICT architecture and concepts describing how <b>Applications, Platforms</b> and <b>Smart systems</b> may collaborate.	
SoA	Service Oriented Architecture. An IT-Architecture where an application system is delivered by exposing, composing, coordinating multiple smaller functional service components.	Web-services

WoT	IoT Architecture defined by WoT Consortium WoT Thing, WoT Consumer, WoT Intermediary, WoT Things Description (TD), WoT Things Directory (TDD)	
RDF	Resource Description Framework. A W3C standard for describing the meta-data needed for data exchange from heterogeneous sources	
SPARQL	SPARQL is an RDF query language—that is, a semantic query language for databases—able to retrieve and manipulate data stored in Resource Description Framework (RDF) format.	

## 2. Overall Architecture and domOS Reference Model

### 2.1. Layered Model

The purpose of the domOS eco-system is to enable future smart energy applications to seamlessly connect to sensing and actuating devices in new or retrofitted buildings. Figure 3 shows a conceptual layered model of the domOS eco-system and its major functional components. The figure also contrasts the domOS layers with a typical IoT stack.

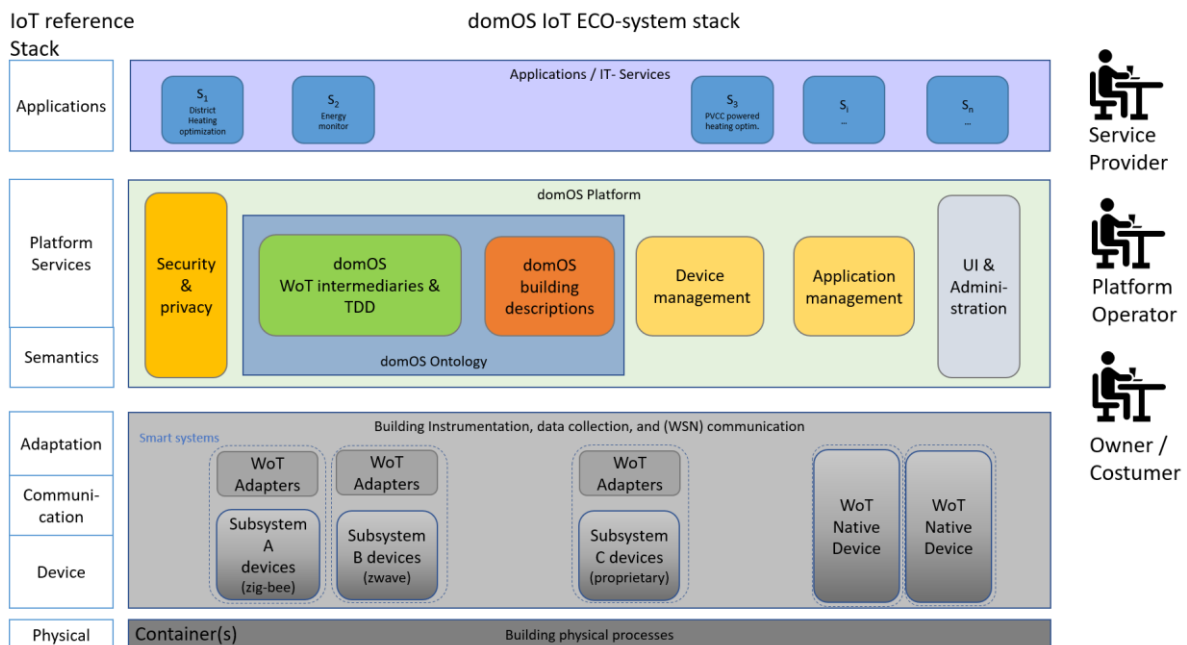


FIGURE 3: LAYERED MODEL OF THE DOMOS IOT ECO-SYSTEM

The lowest layer is the physical structure and processes (e.g., thermodynamics) in the buildings. On top of that is instrumentation by adding sensing and actuating capabilities typically hosted by wireless sensor network devices. To make these WoT compatible, a further abstraction and communication layer known as adaptors may need to be added to connected systems that are not WoT native. According to the terminology (Table 1), these are referred in domOS as “*smart systems*”. When made domOS compliant by adding the necessary semantic annotation, we henceforward refer to them as “*domOS (WoT) things*”.

The next conceptual layer is the platform layer that bridges the gap between smart applications and raw devices by providing a common, secure, and privacy aware semantic interoperability layer.

The upper layer hosts the potential applications using the platform.

**The Service Provider** delivers a business service through an **application**. Applications are registered on the platform along with an application manifest. The service provider may need to approve the application activation for a given building owner, as this may require setting up a billing- and legal-contract which is currently out-of-scope for the platform.

**The building owner (customer)** is the owner of a container – normally its inhabitants. The owner deploys connected systems, activates services, and manages their security properties. For multi-apartment buildings, the housing association may take the role as owner, and then assumes the responsibility to get the required approvals from the tenants.

**The platform operator** is responsible for operating a platform instance on behalf of customers and service providers, i.e., deploys the platform, authenticates, and registers applications to be listed, manages users/owners, and oversees its correct operation.

## 2.2. Main Functional Principles

The main principles behind the platform as derived from the process outlined in Section 1.3 are outlined below.

### 2.2.1. WoT Compliance

- **WoT is used as common device abstractions.** If a device is not WoT compliant, an “adaptor” must be created that facilitates communication to the device using the WoT protocol and provide a suitable TD. The domOS platform does not specify where the adaptors and underlying device drivers are hosted, i.e., whether they are on a gateway, in a cloud, or co-located with the platform or not. The things description should also be made compliant by semantically annotating it using the domOS ontology.
- A **thing description (TD)** (W3C (WoT TD), 2020) supplied by the smart system manufacturer (or other external party) is registered in the **Things Description Directory (TDD)** by the building owner using suitable user interfaces when deploying the smart system device. The W3C WoT (W3C (WoT TD), 2020) recommends the semantic annotation of TDs using semantic resources. We recommend the semantic annotation of thing descriptions using domOS Common Ontology (dCO), available at <https://w3id.org/dco>, as unique source of semantic truth for domOS project. A thing and its interaction affordance (properties, actions and events) can be annotated using the “@Type” field (e.g., “@type”: “dco:temperatureSensor”, “@type”: “dco:humidityPropertyAffordance”). The units of measurements of the interaction affordance can be annotated using dCO (e.g., “unit”: “dco:Celsius”).
- **A running application is a WoT consumer.** An application may have an internal structure (e.g., SoA micro-services) but is a single artefact, as seen from the platform perspective.
- **A WoT Intermediary** may act as a proxy that decouples applications and smart systems by **hiding WoT device credentials from applications**. It may **check that every access** to a property, device, or affordance has been granted permission by the building owner.

### 2.2.2. Building Description

- A **Building Description** is a description that defines an abstract description of the building and its properties as needed by the smart services. It consists of
  - a description of building specific metadata (e.g., total space, thermal capacity, envelope type),



- a description of the structure of the building (e.g., floors, apartments, rooms),
  - internal building tasks (e.g., dco:spaceHeatingTask): The metadata of tasks are defined along with the list of the involved devices,
  - the static properties of the spaces (e.g., thermal capacities, space),
  - the list of the device types that are installed and supported by the building,
  - possibly further static properties of the devices, (e.g., heating surface areas, maximum on time, and their capacities),
  - a semantically enhanced description of the underlying “things”, and their semantic relation to and dependencies on other “things”, and
  - a link to the “Things” description needed to interact with the underlying physical device
- The building description properties are defined by the **domOS Common Ontology (dCO)**, and the schemas needed to process it is derived from that.

### 2.2.3. Device (Things) Management

A platform may need to support management of things:

- The platform may offer support for management of the **life cycle of the Things**: commissioning, decommissioning, pausing, replacing, etc.
- **Health Monitoring** of devices enabling fault tolerant operation (fall back to a reduced service level) through user notifications and notifications to involved services. The failure detection can be basic connectivity checks via heartbeat or probing mechanisms, or advanced functional checks, diagnosis, and anomaly detection.

### 2.2.4. Application Management

A platform may need to support management of applications:

- Lifecycle management of applications: Subscription and activation, pausing, termination, etc.
- Monitoring and failure detection (connectivity checks via heartbeat or probing mechanisms) enabling failure notifications and fallback to local control loop.
- **Authentication**. Only approved users operate the platform and its managed containers. Only approved applications appear on the platform.
- **Compatibility check**: the platform prevents launching an application that requires features (measuring and control points) that are not provisioned in the building. We remark that checking the correct operation of an application (or set thereof) is untrivial (Le Guilly, 2016) (Pedersen, 2018) and beyond the responsibility of the eco-system.
- **Compatibility check**: the platform enables application to check that the building is compatible with it by enabling it to check whether the building is equipped with the required set of measuring and control points for that application. If not, it will go into operational mode, and quit after notifying the user.
- **Feature interactions and correctness**: the platform may give a warning if the user tries to activate an application that writes to a property that is already potentially written by another

application. We remark that this is insufficient to guard against undesired feature interactions<sup>1</sup> (Le Guilly, et al., 2016), but nevertheless a useful heuristic. We also remark that checking the correct operation of an application (or set thereof) is untrivial (Le Guilly, 2016) (Pedersen, 2018) and beyond the responsibility of the eco-system.

### 2.2.5. Security and Privacy

There are several facets of security and privacy.

- **Information hiding:** Only the necessary information about a container’s building description is revealed to applications, and users must explicitly grant the necessary access rights to the requested properties.
- **Transparency:** The owner is provided with a comprehensible view of exactly which building properties and connected systems that an application has access to, and in what mode.
- **Authentication:** Access to a container and devices is only provided by authenticated and authorised applications.
- A platform may support **multi-tenancy**, i.e., ensure that the information related to multiple buildings and owners are kept isolated.
- An **access control mechanism** is needed. It can for example be based on the building owner granting simple “Read/Write” permissions. This can be done on different granularities, e.g., container, things, or WoT property levels and be granted on a per application basis through specification of an access control matrix. For example, to specify access control on affordance level, an annotation is required to indicate:
  - Calling an action affordance requires both R+W rights
  - Traversing a link- affordance requires R rights
  - An interaction affordance can be **read** by an application if it has been granted R permissions
  - An interaction affordance can be **written** by an application if it has been granted W permissions
- **Control:** Any access to the smart systems is under control of the platform.

### 2.2.6. User- and Administrative Interfaces

- The end-user facing user interfaces are primarily web-based (“web-apps”) graphical user interfaces allowing “ordinary” people to manage their container, devices, and subscriptions.
- Other interfaces may allow import of RDF-data sources.

---

<sup>1</sup> An example of undesired feature interaction that cannot be detected by overlapping writes is a heating controller that tries to keep temperature, while an indoor climate application opens the windows to reduce humidity. A second example is an HVAC application that opens windows whose movement may trigger the alarm systems motion sensors.

The defined platform is defined functionally, that is, it does not specify where its components are deployed (i.e., on an existing or dedicated gateway/edge, cloud, or combination thereof), similarly no extra-functional requirements are specified.

### 2.3. domOS Eco-System (Normative)

As elaborated in Section 1.1.5, the domOS ecosystem functional specification must be flexible and tailorable to deployment flexible to handle small installations and large installations in different application areas.

We therefore classify the list of suggested features as either being a **mandatory, recommended, or optional** feature in a platform instance. Furthermore, the detailed specification and behaviour of a feature may be defined by domOS (so-called core feature) or be defined the platform. Following this principle whilst respecting the eco-system requirements, the domOS ecosystem is specified by Table 2.

TABLE 2: DOMOS ECO-SYSTEM SPECIFICATION

Feature	Level (M) Mandatory (R) Recommended (O) Optional	Specification Responsible Domos Core, or Platform instance	Note
Devices as DCO annotated WoT Things	M	Core	1)
Applications as WoT Consumers	M	Core	
Building Descriptions using DCO	M	Core	2)
WoT TDD	R	Core	2)
WoT Intermediaries	R	Instance	3)
Forced WoT intermediation	R for closed platform M for open platform	Instance	
Privacy Rules, Information Hiding	M	Instance	4)
Transparent Security and Privacy status view	R	Instance	
Access-control	R M for open platform	Instances	
Application Authentication and Authorization	R	Core	5)
Platform multi-tenancy	R	Instance	
Device management and monitoring	O	Instance	
Application subscription and lifecycle management	O	Instance	
Application compatibility check and feature interaction check	O	Instance	
User- and- Administrative Interfaces	O	Instance	

- 1) DomOS Common Ontology
- 2) For a WoT consumer (application) to function, it needs to retrieve the TDs (things descriptions) for the devices it interacts with. Whilst it is technically possible to build a WoT consumer by retrieving the TD from elsewhere, e.g., by embedding it in the building description or loading from a file, doing so is not generally the best way. The strongly recommended general way is to retrieve the TD by following the WoT discovery (W3C (WoT Discovery), 2021) specification and



serve TDs via a TDD (things description directory) to enable a standardized, scalable, maintainable, and searchable device discovery mechanism. Access is illustrated in Figure 9.

- 3) Whilst the Wot Intermediaries are not mandated, it is a useful mechanism to solve several principal problems.
  - a. Forced intermediation
  - b. Enforce access control
  - c. Device abstraction, aggregation, and local processing
  - d. Caching and storing last known values of “things”

Access via an Intermediary enforcing access control is illustrated in Figure 8.

- 4) A platform instance must specify how it deals with security and privacy. The mechanisms and policies may vary among instances.
- 5) If authentication needs to be performed, the platform is required to specify how this Authentication and Identity management is done. Currently the widely used and industrially accepted is to be OAUTH2. Hence, this is the current recommendation for domOS.

### 3. Platform Instantiation Guidelines (Non-normative)

This section elaborate on how a larger platform may be structured and behave. It provides suggestions about structure, behaviour, and possible implementation technologies.

#### 3.1. Use-Case Diagram

The main use-cases for the domOS platform are captured in Figure 4. To keep focus on essential behaviour, it excludes simple use-cases for creating, reading, updating, and deleting registered data in the platform.

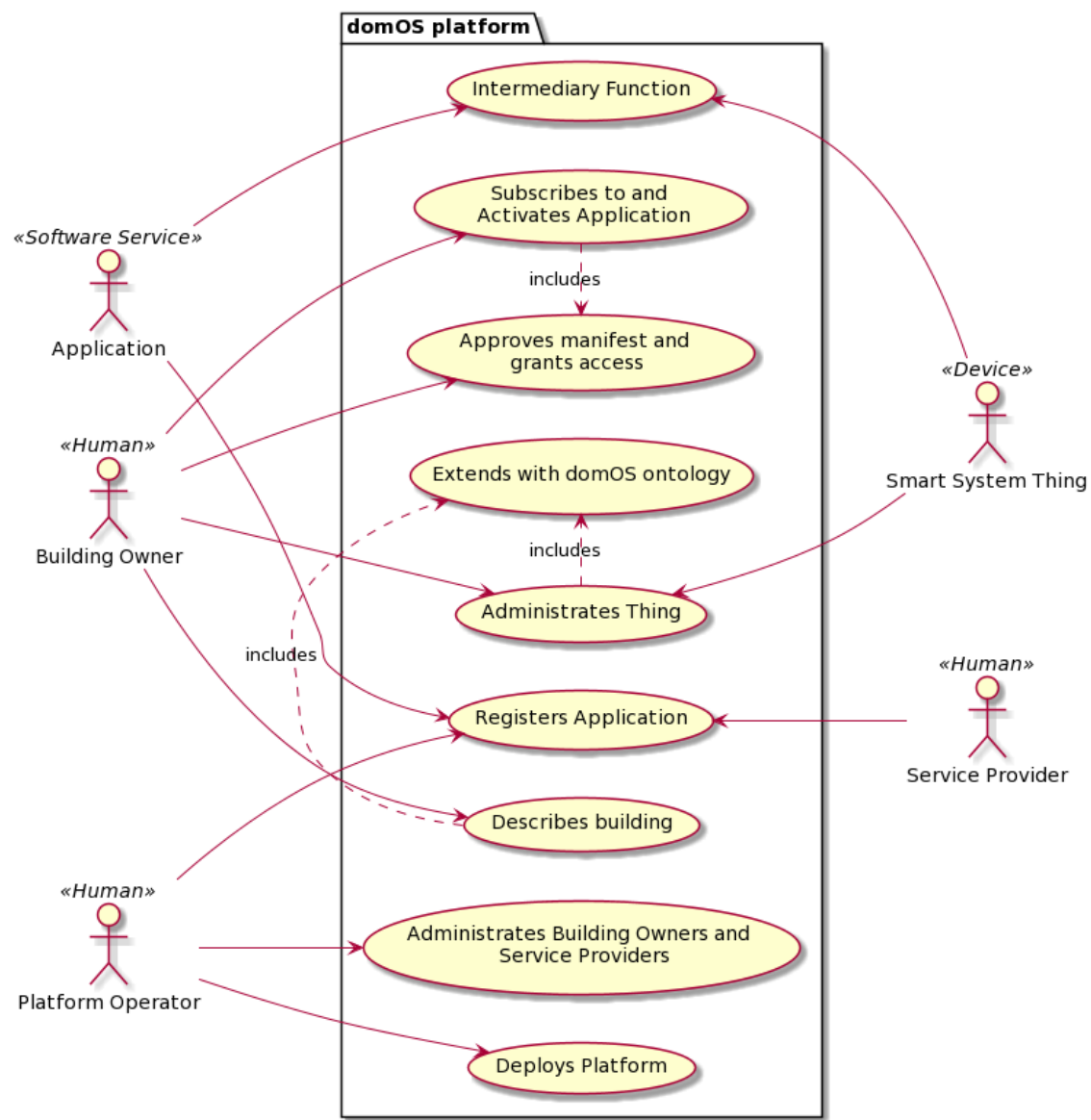


FIGURE 4: HIGH-LEVEL USE-CASES FOR THE DOMOS PLATFORM

### 3.1.1. Use-Cases

#### Register application

- The platform operator receives a request from the service provider to list an application on the platform.
- The operator registers an application as available in the platform. This includes registering an application description, the manifest, an activation endpoint (e.g., an API URL, or a web-hook) and security credentials that the platform requires to be able to activate the application. The necessary information is received out-of-band from the service provider.

#### Subscribing and activating application

- The building owner browses the potential applications.
- Attempting to activate presents the application's manifest that owner needs to approve.
- Once also the service provider has agreed to the activation, the application receives the building description from the Building Description Directory (BDD) and explores the provided information about the building and deployed devices. This includes the number and types of the devices and spaces. Only the parts of the building description - as filtered by the manifest declarations - is returned to the application.
- The application checks that the required set of devices and interaction affordances are available.
- The application requests access to the required list of specific things and specific affordances. A request is automatically denied if the requested device type is not listed in the application manifest.
- The user may auto-approve the access to things that matches the manifest, or the user may choose to review details (specific devices and access mode); The owner approves the request (all-or-nothing) as we assume that all requested properties are necessary for the correct functioning of the application.
- Once approved the application starts functioning.

#### Intermediary function

- The application requests the TD it needs.
- The platform returns the TD for the things intermediary.
- The application accesses an affordance on the thing.
- The intermediary gets the request, checks permissions, and relays the request (and response) to (and from) the underlying thing.
- Bypassing the intermediary function is only possible when the manifest has added a "RAW ACCESS" annotation to the device type, and the user has approved this provided suitable warning. The platform returns the TD for the smart thing.

#### Registering/Updating TD

- The building Owner prepares and annotates the Thing Description of the things it wants to register or update.



- The TD is submitted to the TDD and saved after a validation.
- A TD can be updated and removed from the TDD.

#### Building modelling

- Using the domOS common ontology, the building owner describes the building structure (division into floors, rooms, and spaces), its metadata, and existing IoT things devices.
- The owner annotates domOS device types (as defined in the ontology) in the appropriate building spaces.
- The building owner provides a link (access URL) between the device type and the actual domOS thing that supplies the actual functionality of the specified device type. The installed things are assumed to be registered in the TDD.

## 3.2. System Structure

### 3.2.1. Components

The Platform will need multiple components. For management, the Platform operator will need the ability to manage the permitted Applications and to manage Customers. The Customers will need to manage their Container by integrating Smart Things and adding semantic information. The Platform will contain components to handle proxying for Application initiated interactions with WoT devices, but also a component to handle communications initiated by the device such as events. Various credentials will need to be handled carefully, including the ability to verify Applications, Components, Customers, and the Customer will need to add to its Container the necessary credentials to interact with the WoT devices.

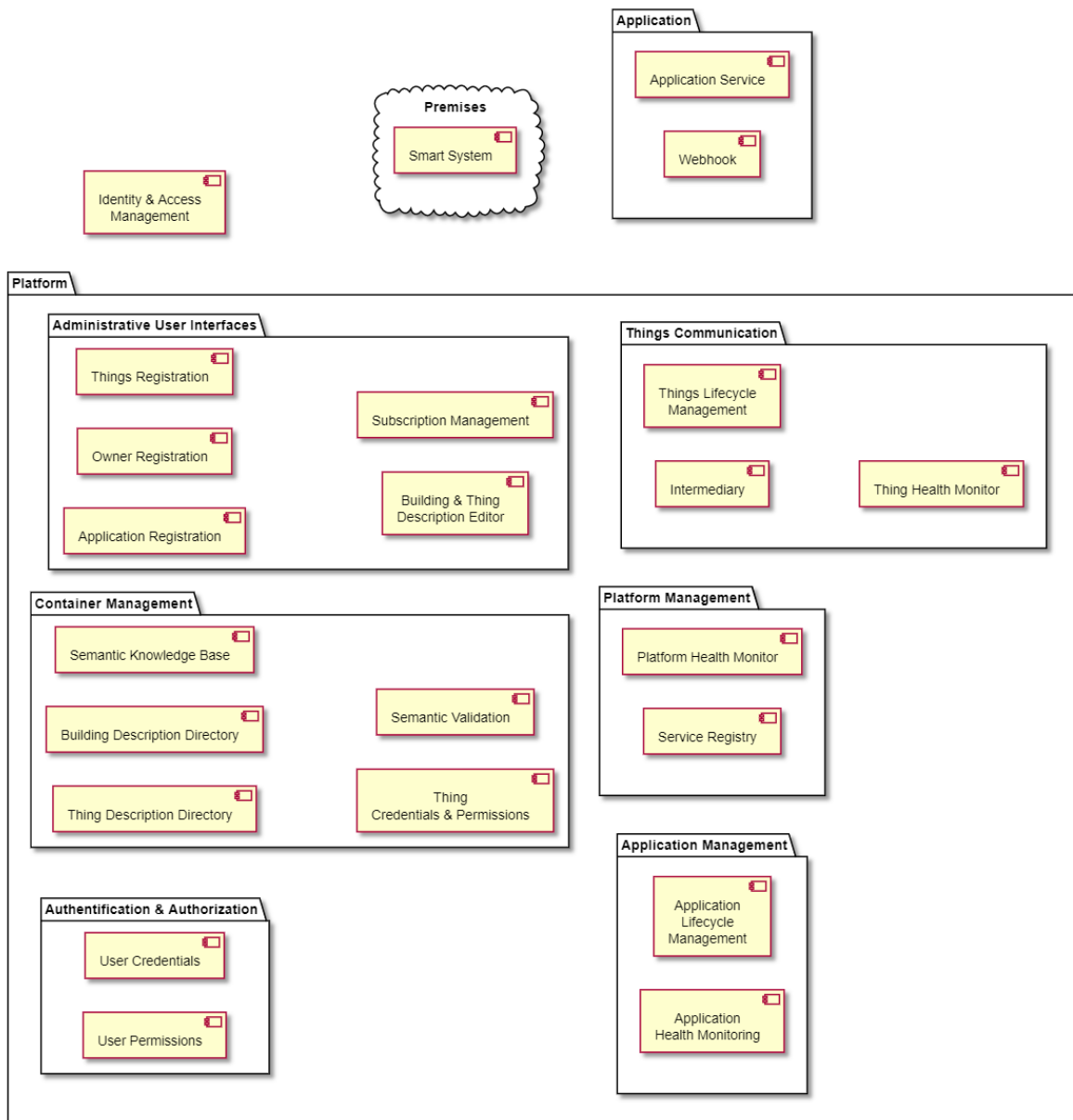


FIGURE 5: DOMOS COMPONENT DIAGRAM

The following paragraphs define functional responsibilities of each component.

### 3.2.1.1. Container Management

The components in the Container Management package implement the functions and storage that is logically needed for each container.

#### TDD - Things Description Directory

- Stores things descriptions (possibly in the Semantic Knowledge Base).
- Provides access to the Thing Descriptions stored in the Things Directory by serving the API requests (serves HTTP requests, parses request, routing, etc.)

- Serves semantic search for Thing Descriptions and their metadata from the Semantic Knowledge Base.
- CRUD functionality for things (as suggested by to the WoT Specification (W3C (WoT Discovery), 2021)) <https://www.w3.org/TR/wot-discovery/#exploration-directory-api-registration>

#### BDD – Building Description Directory

- Similar functionality as “Things Description Directory” but provides access to the Building Description stored in the Building Description Directory (possibly using the Semantic Knowledge Base).
- Ensures semantic validity of the building description using the Semantic Validation component: e.g., ensure that Things are placed in spaces actually defined by the structure of the building, and that all required semantic annotations are completed.
- Serves semantic search of Building properties.
- Stores temporary building descriptions being created by the building editor.
- Serves Semantic search of Building- (and consequently Things-) descriptions based on the Semantic Knowledge Base.

#### Semantic Validation

- Semantic search allows WoT consumers to search for metadata and IoT devices without having any prior knowledge about them. The semantic search is based on contextual information in a smart building.
- Semantic Validation supports the Container Owner, via the Building and Thing Description Editor, to augment his Building Description with the necessary semantics.
- Semantic Search provides more intelligent searching than just reading collections of semantic data. (Thing Descriptions and Building Descriptions can be stored in the Semantic Knowledge Base to allow more advanced semantic search).
- Semantic validation aims also to semantically validate Thing Descriptions and Building Descriptions, making sure that they do not include inconsistencies. For instance, a thing description of a temperature sensor cannot measure the power property.

#### Semantic Knowledge Base

- Provides one source of semantic truths. The semantic knowledge base stores the RDF instantiation of Thing Descriptions and Building Descriptions.
- It is also thought of as an abstraction to gather useful query logic to avoid duplication in other services, and also to separate the underlying implementation from the actual query needs.

#### Thing Credentials & Permissions

- Stores the actual credentials to be used by the Intermediary to fulfil an Application interaction.
- It also stores permissions for which user (application) ids may access which Thing affordance.

### 3.2.1.2. Security - Authentication, and Authorization

To support the necessary security and privacy requirements, the platform must authenticate all requests and all communication must be done over authenticated encryption (e.g., TLS/HTTPS). Each user on the platform, which includes platform operator, applications, container owners, and internal platform services have an ID. In addition, depending on how the user contacts the platform, they may have other information. In particular, with respect to applications and the IAM (using OAuth2):

- Applications and components have a secret,
- Platform administrators and container owners may have username and password, also another factor of authentication (2-factor).

The particular mechanism used to authenticate components internal to the platform is left unspecified. Component as mentioned could also be users in the IAM component authenticated by the JWT in their access key, or components could be preconfigured with digital certificates.

Security checks are pervasive in the platform, and we consider checks that happens on (a) any requests, (b) when a particular operation is done in the platform, and (c) when an application service interacts with WoT devices via the intermediary component.

- For any request to even be considered for processing in the platform, the sender must be authenticated and present a valid access key. The only exceptions are the core functionality of the IAM (Identity and Access Management) component to acquire an access key, and the Platform Management service (which can tell you how to find the IAM and other components). Access keys are acquired by contacting the IAM with the ID and the extra information and may have a limited time of validity upon which another has to be reacquired or refreshed.
- With a valid access key, users' requests are now processed by the platform. Components (including the User Permissions component itself), check with the User Permissions component that the request is to be permitted, before processing it.
- Request containing WoT interactions with the platform's Intermediary component are also checked against the Thing Credentials & Permissions component before being processed.

### 3.2.1.3. Things Communication

#### Things Lifecycle Management

- Supports the planned and unplanned replacement of Things in a Building to minimize disruptions to application services by allowing a new device to replace an old one with similar capability. The lifecycle states are also elaborated in the WoT draft specification (W3C (WoT Architecture 1.1), 2020).

#### Thing Health Monitor

- Monitors defined devices to see if connection can be established and reports non-recoverable errors to the building owner and enables applications using the device to discover the presence of errors.



- It could use information from the Intermediary component about its recent communications.

#### Intermediary

- Act as an WoT intermediary and is the entry point for all Application service interactions with WoT devices using WoT communication. This component separates Applications from Containers.

#### 3.2.1.4. Application Management

##### Application Health Management

- Checks that a subscribed application is alive and responsive by implementing a suitable failure detection mechanism, typically by monitoring the communication from the application, probing it by calling its web-hook, or by detecting missing “heart-beats” from the application. If it is suspected that the application cannot reach the platform, or vice versa, for a prolonged period, and automated recovery attempts are unsuccessful, the owners, application provider, and platform operators are alerted.

##### Application Lifecycle Monitoring

- Similar to things, an application also has a lifecycle where the status of the application changes during lifetime (“onboarding”, “subscribed”, “active”, “paused”, “unsubscribed”, “decommissioned”, etc.). The platform may need to react to these changes, e.g., if the application is updated, it may need to access further devices, and hence need the user to re-approve the application.

#### 3.2.1.5. Platform Management

A platform implementation may consist of multiple components that are potentially deployed on different devices (e.g., building gateway(s) and a cloud). Therefore, the platform should have functionalities to maintain the deployment of the platform itself, i.e., register and maintain the location and execution status of its (potentially) distributed components.

##### Platform Health Monitor

- Monitors the necessary components for uptime and errors of the platform to prevent interruptions. Notifies platform operators such that malfunctions can be remedied immediately.

##### Service Registry

- To support the various deployment possibilities of the platform, this component act as the registry where components can query for the information on how to reach other components. Might also contain public keys to authenticate those components.

#### 3.2.1.6. Administrative Interfaces

The administrative interfaces contain the graphical (web-based) user interface (owners, operators, application providers) functionality to register and manage things, register, and manage owners, register,



and manage applications. It also enables owners to construct building descriptions, and to subscribe to applications.

#### Things Registration

- Enables the activation/deactivation of an already described Thing.
- Provides user interface for administrating things credentials
- Provides user-interface for managing the life cycle of the Thing.

#### Owner Registration

- Registration and editing of Building owners' personal data, and owned container(s).
- User account management

#### Application Registration

- Applications undergoes a registration process by the platform operator.

#### Subscription Manager

- Supports building owners in subscribing approved applications to their container.
- Keep track of the current subscription of applications to building owners.

#### Building & Thing Description Editor

- Enables the insertion of a building description for a container.
- Optionally, supports iterative modelling of the building description with a graphical interface.
- Provides detailed information about semantic validity of the building description to ensure compliance with the ontology.
- Provides the owner with an overview of the privacy status of his building, i.e., which applications are active and their access rights (which things and affordances are accessed by the application in what R/W-mode).

### 3.2.2. Main Interfaces

Interactions with WoT devices works under the WoT architecture where applications, as consumers, interact with things, with the platform as an intermediary. If the application has not already identified the interaction affordances previously during subscription activation, it can, using the TDD (Things Description Directory), and the BDD (Building Description Directory), query for Things to interact with using the Semantic Search capability. When receiving a WoT interaction, the Intermediary component checks whether the Interaction is allowed and retrieves the credentials necessary to authenticate with the WoT device. Then it performs the interaction with the thing with itself as the consumer and relays the response back to the application.

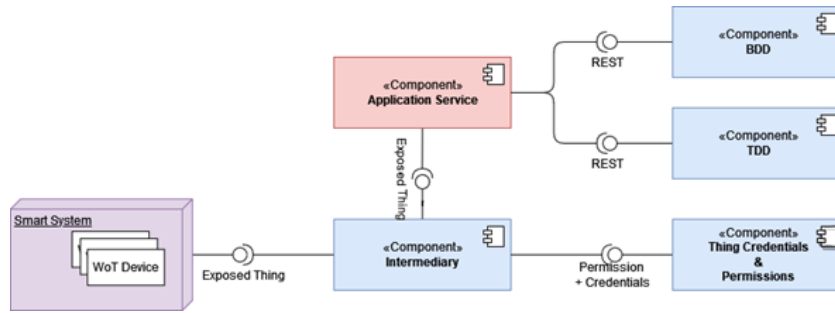


FIGURE 6: COMPONENTS INVOLVED IN WoT INTERACTIONS

### 3.3. Behaviour

#### 3.3.1. Adding a WoT Device

We consider a simple scenario where the building owner uploads a TD via a GUI “upload thing”. We assume that the building owner is authenticated on the platform and has a valid active session. To deploy a WoT device into the platform, the device must be compatible with domOS (See Section 2.2.1) and a Thing description should be provided. The thing description should contain semantic annotations from the domOS ontology. After adding the thing description to the TDD via the editor, a compliance check can be done to show inconsistencies with respect to the ontology, and also to notify about missing credentials for the thing that would prevent interaction with the device. A full UI could include a GUI that helps the user construct the TD interactively.

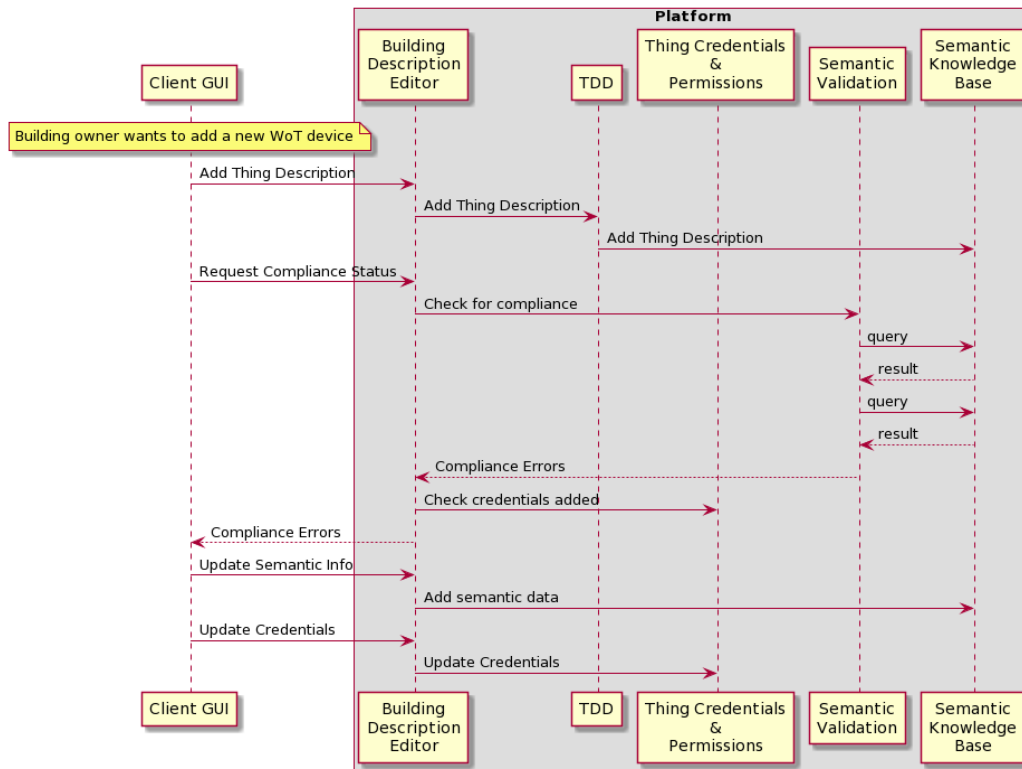


FIGURE 7: ADDING A NEW THING DESCRIPTION

### 3.3.2. Normal Scenario Where an Application Accesses (reads) a WoT Property

#### 3.3.2.1. Open Platform

In an open platform, some conditions must be met before an application can read a WoT property. First, the application needs to be registered on the domOS platform (ref to registering an application). Secondly, the customer owning the device from which the application wants to read the property must have subscribed to that application (ref to subscribing to an application). The interaction is performed in a 3 steps process, with the first two performed only when application needs to refresh its authorization token or when it doesn't have the Thing Description of the WoT Thing it wants to interact with. In the first case, the application sends a request to the intermediary, which is proxying the Smart Thing. In this scenario, the request is denied because the application needs to refresh its access key. The application then sends a refresh access key request to the IAM which delivers a valid access key.

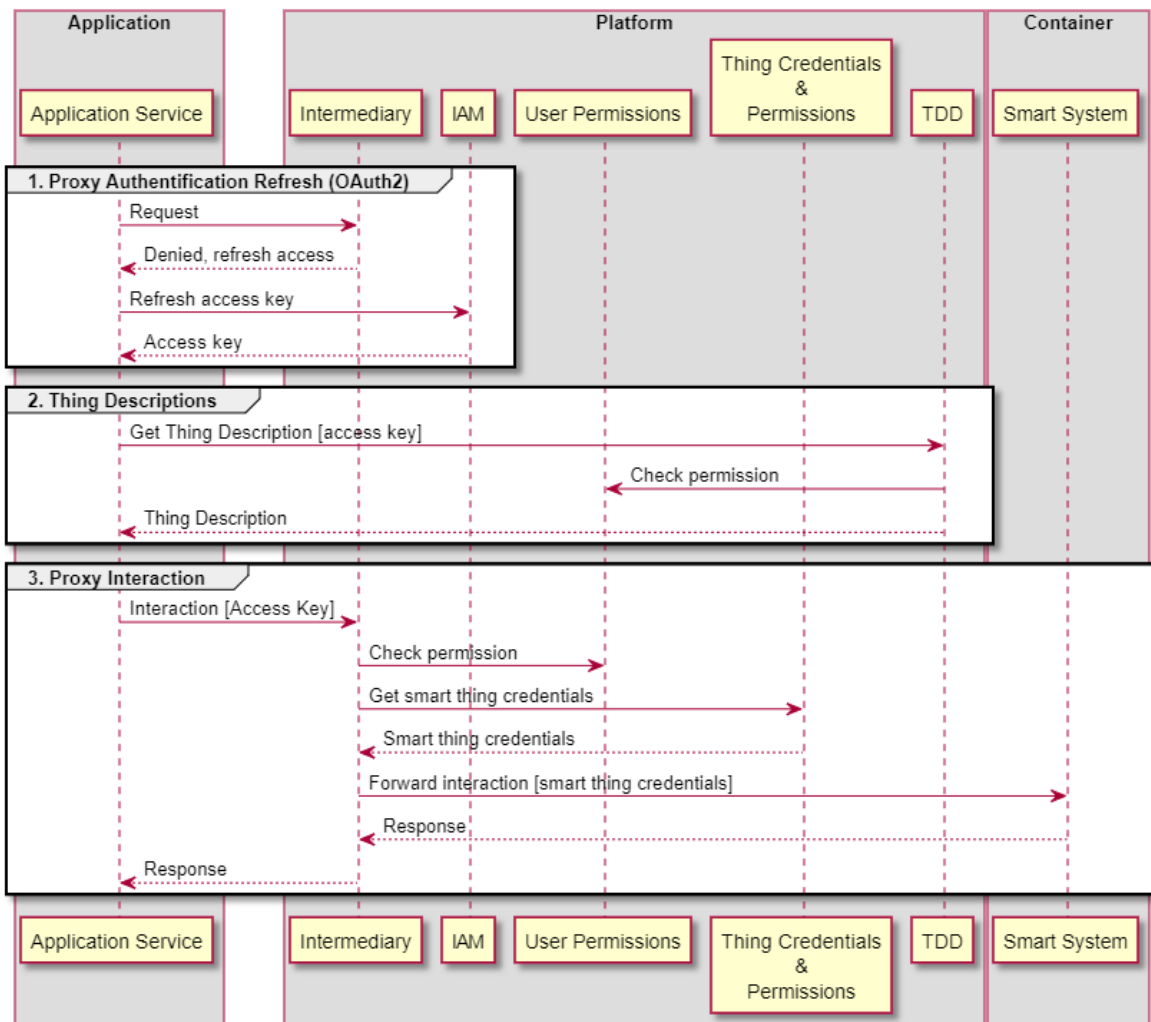


FIGURE 8: APPLICATION ACCESSING A WoT PROPERTY AFFORDANCE IN AN OPEN PLATFORM

Then, using the access key obtained from the IAM, the application interacts with the Thing Description Directory to get the Thing Description of the Thing it wants to interact with. The third step is the actual interaction with the WoT Thing. The application sends the interaction request along with its access key and the id of the Thing to the proxied Smart Thing based on the information from the TD. Permission check is performed by the Intermediary with the User Permission Component. If the application is allowed to perform the interaction, the Intermediary will request the WoT Thing credentials to interact with the real thing. Having the credentials, the intermediary forwards the interaction to the Smart System, and when getting response from the WoT thing, forwards it to the application.

### 3.3.2.2. Closed Platform

For the case of a closed platform, two possible reading options: getting information from a TDD or from a BDD. The application requests the thing description of the Thing it wants to interact with from the TDD. With the TD, it can interact directly with the thing. The second option is to go through the BDD and get the building description. With the BD, the application can interact directly with the thing.

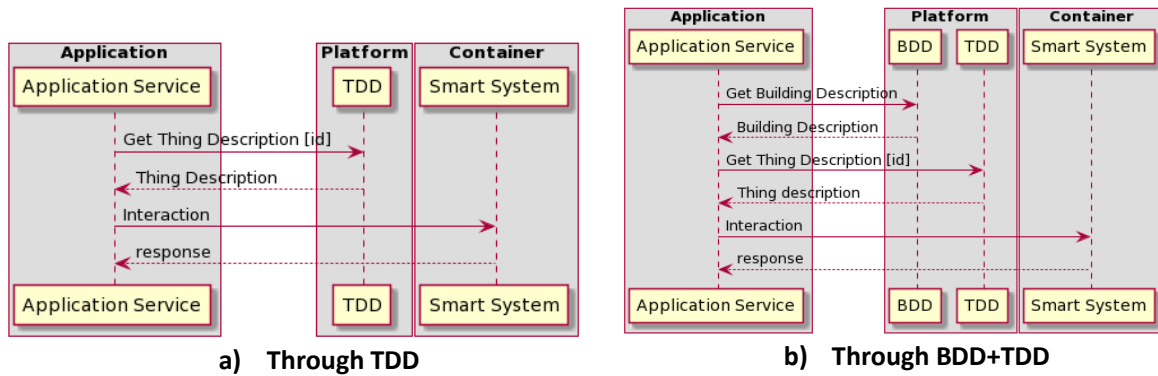


FIGURE 9: APPLICATION ACCESSING A WoT PROPERTY IN A CLOSED PLATFORM

### 3.3.3. An Event Occurs in a WoT Device

Reading, writing, or invoking actions on a WoT device can be done by forwarding the request to the device (assuming valid permissions). Another method of interacting with a WoT device is to passively receive information on changes, either by subscribing to events from the device, or by observing observable properties. In this case, the request cannot simply be forwarded. Acting as a WoT intermediate, when an application request to subscribe to an event or property, the Platform will need to create the subscription to the actual device and act as the intermediary. Figure 10 demonstrates an Application requesting to listen for a particular event. The Platform checks whether it has permission to do so, starts actively listening to events from the WoT device, and forwards all events to the Application.

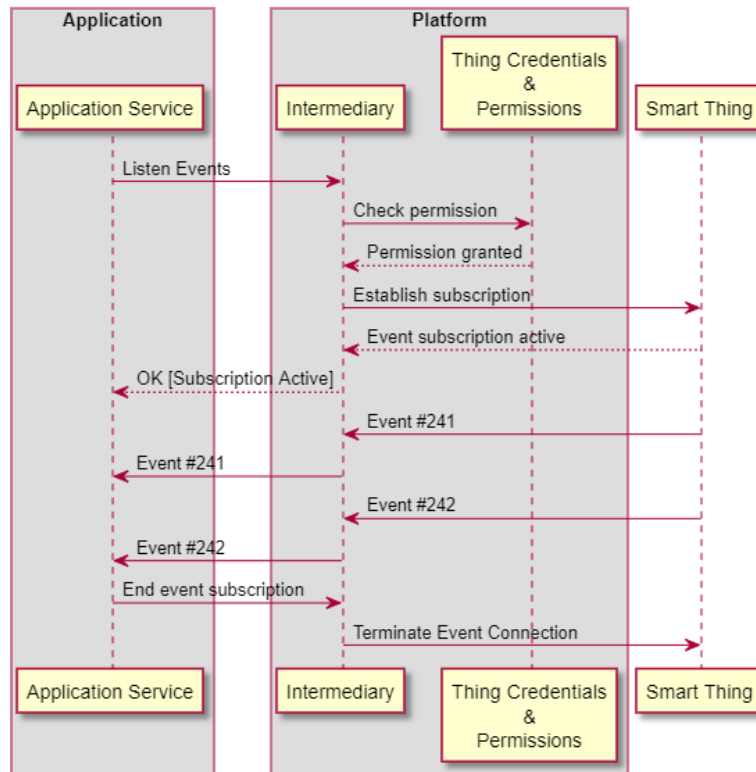


FIGURE 10: APPLICATION LISTENING TO EVENTS

### 3.4. IoT Semantic Interoperability Using Semantic Web Technologies

Semantic web-technologies and tools can be helpful when implementing system components that operate on semantic descriptions, such as things- and building-descriptions.

Semantic heterogeneity of an IoT ecosystem presents a major bottleneck to implement IoT applications. Service providers and IoT developers cannot be aware of the existing available devices in the domOS IoT ecosystem. We developed domOS Common Ontology (dCO) as a common vocabulary to achieve semantic interoperability in smart buildings for the WoT ecosystem. The dCO is available at <https://w3id.org/dco> and is the result of a collaboration between academic and industrial partners from different backgrounds. IoT semantic interoperability allows humans and machines to have a common understanding of the domOS ecosystem. After interviewing partners from the five demonstration sites, we have identified the following use cases of the dCO:

- Semantic annotation of Thing Descriptions.
- Elaboration of Building Descriptions.
- Elaboration of Manifest files.
- Semantic validation of Thing Descriptions and Building Descriptions.
- Semantic search of metadata from the Semantic Knowledge Base.
- Compliance check of new IoT applications in a smart building.

Service providers and IoT developers cannot be aware of the existing available devices in the domOS IoT ecosystem. It may be useful to use a semantic search capability that enables IoT consumers to search for the required IoT devices. For instance, a space heating service provider wants to know the list of IoT sensors in a given room.

One of the main challenges of implementing semantic interoperability in a WoT ecosystem is to enable IoT developers to search for IoT devices or other metadata information without having any prior knowledge about them. To enable semantic search of IoT devices, Thing Descriptions and Building Description are stored in a Semantic Knowledge Base. The semantic search process enables the search for IoT devices, building descriptions based on their context such as the location, the type of the device, the related services, and any other contextual criteria. In the Figure 11, we present the semantic search sequence diagram. A client (here a platform software component) can send semantic queries to search for some metadata, such as the list temperature sensors in an apartment, the thermal capacity of a building, and the space of a building. The query access management component verifies if the client has the right to access it's required metadata. This can be done by filtering the queries based on the access rights of the client. For instance, if the client is the building owner, he can access all the metadata. If the client is a service provider, he can access only to the metadata where his application is provided.

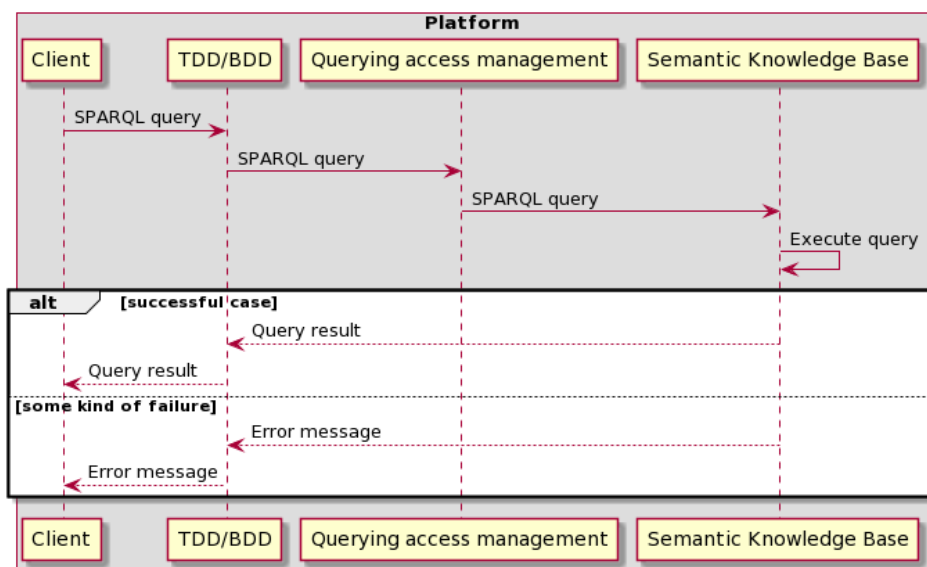


FIGURE 11: SEMANTIC SEARCH SEQUENCE DIAGRAM

Another challenge to confront in IoT ecosystems is allowing service providers to provide application to users based on the heterogeneous existing IoT infrastructure in smart buildings. Service providers do not know if their applications are compliant or not with a given IoT platform. Also, households do not know the compliant applications for their home. For instance, a space heating service provider would like to implement his application in a smart building. He should first know if the building contains the set of IoT devices (e.g., heater, temperature sensor) required by his service. To solve this issue, we propose an IoT service compliance check process.

In Figure 12, we present a sequence diagram exemplifying how semantic search could help in the compliance check process. The process begins by a client component requesting to check its compliance for the IoT platform via the TDD/BDD APIs. The API forward the manifest file and the application id to a SPARQL query generation component. This component generates a semantic query (SPARQL query) based on the manifest file, the application id, and the access rights of the client. The manifest file contains all the required IoT devices to run a given application. This manifest file is made by the service provider of the application. The generated SPARQL query is executed in the Semantic Knowledge Base. If the execution is successful, the API return a message to the application. This message specifies if the application is compliant or not.

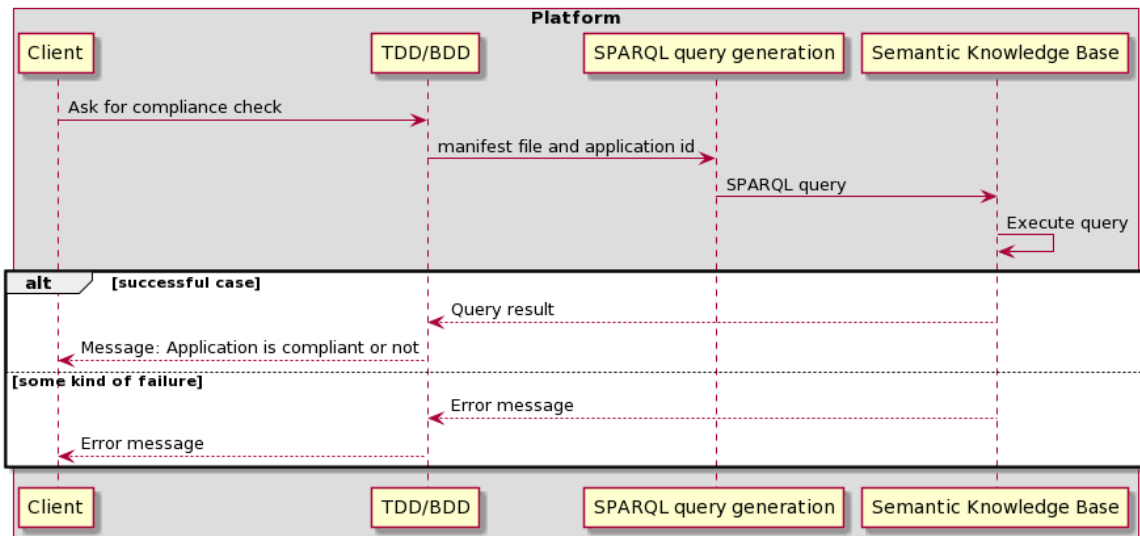


FIGURE 12: COMPLIANCE CHECK SEQUENCE DIAGRAM

## 4. Key Interface Specifications (Normative)

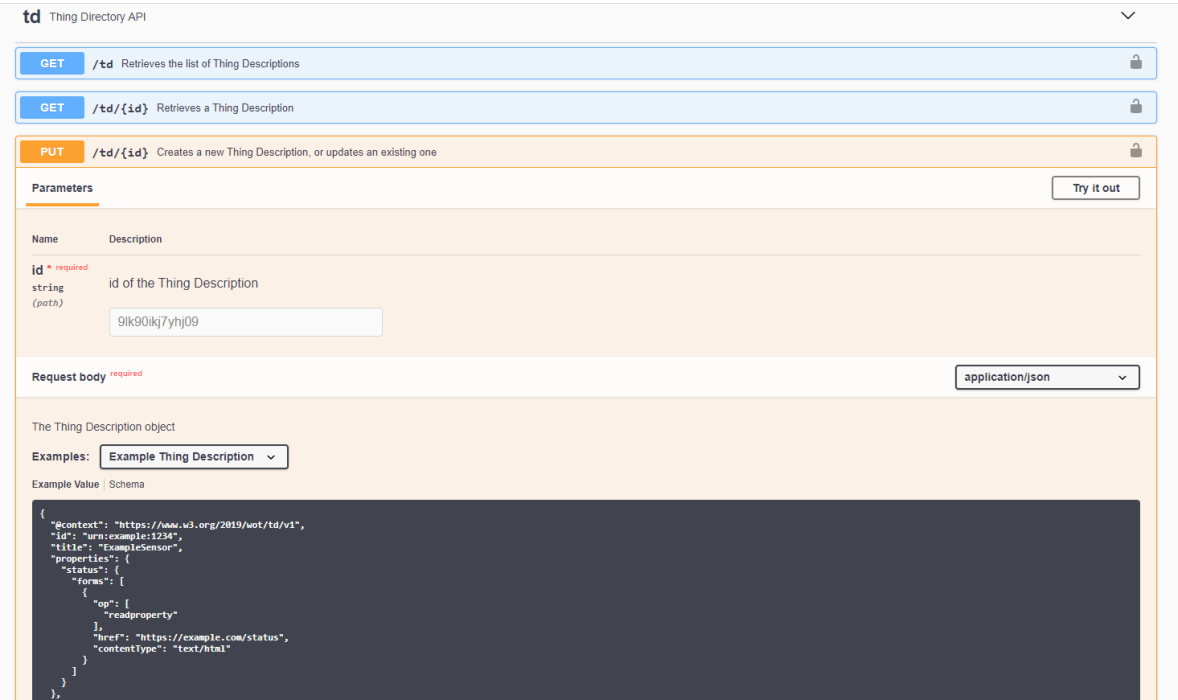
### 4.1.1. Swagger Key Interfaces Specifications

Swagger (SmartBear Software) is a framework for designing, building, and documenting REST-based APIs. For the API-specification, Swagger proposes to use the OpenAPI Specification which defines a standard (how to define parameters, responses, paths, models, etc), language-agnostic interface to RESTful APIs. It allows a programmer to understand what the API does and how to interact with its various resources. OpenAPI specifications are written in JSON or YAML format and can be used with the Swagger UI for interactive documentation (See Figure 13).

One advantage of using Swagger for API specification is the various possibilities offered such as the automatic building of highly readable and interactive API documentation, the automatic generation of client libraries and server skeleton for the API in many languages, and finally, test generation and automation.

The different APIs, names, and parameters are provisional and will evolve throughout the development of the domOS platform. Inputs and outputs will be further refined and concretized as the platform API evolves.

The API specifications are hosted at: <https://herald.aau.dk/docs/>



```
{
  "@context": "https://www.w3.org/2019/wot/td/v1",
  "id": "urn:example:234",
  "title": "ExampleSensor",
  "properties": {
    "status": {
      "status": {
        "forms": [
          {
            "op": [
              "readproperty"
            ],
            "href": "https://example.com/status",
            "contentType": "text/html"
          }
        ]
      }
    }
  },
  "security": [

```

FIGURE 13: EXAMPLE SWAGGER INTERFACE



#### 4.1.2. TDD API

The API specification for the Thing Description Directory defines methods for Thing Description management.

Methods	Resource	Description
GET	/td	Retrieve the list of Thing descriptions <ul style="list-style-type: none"> <li>Output: List of thing descriptions registered which the user has access (JSON-LD)</li> </ul>
GET	/td/{id}	Retrieve a specific Thing description. <ul style="list-style-type: none"> <li>Input: Identifier of the thing</li> <li>Output: thing's thing description (JSON-LD)</li> </ul>
POST	/td/	Register a Thing description Output Identifier of the thing and confirmation
PUT	/td/{id}	Updating a Thing description <ul style="list-style-type: none"> <li>Input: Identifier of the thing.</li> <li>Output: confirmation</li> </ul>
DELETE	/td/{id}	Remove a thing description. <ul style="list-style-type: none"> <li>Input: Identifier of the thing</li> <li>Output: removal confirmation</li> </ul>

The domOS TDD API follows the (working draft) specification (W3C (WoT Discovery), 2021). This contains the minimum procedures that must be implemented by the domOS TDD. The specification plans additional procedures for device management (currently unspecified), events, and semantic search using SPARQL, XPath, and JSONPath.

Thing IDs are assigned by the TDD upon creation. Following the recommendation, it should use UUIDv4 for this (Universally Unique Identifier) (The Internet Society, 2005)

For domOS, semantic search mechanisms are provided through the Building Description Directory component, that allows a filtered semantic description of the building model to be returned to external applications. For domOS, the TDD (nor Building Description) should not allow full SPARQL queries to be executed by external applications for security and privacy requirements.

#### 4.1.3. Building Description Directory

Method	Resource	Description
GET	/bdd/{containerId}	Retrieve the (filtered) building description,
PUT	/bdd/{containerId}	Update the building description associated with the container.
POST	/bdd/compliance/{containerId}	Check a building description for errors, consistency, and compliance with the ontology.

Data that pertains to the building description, but is not part of it, goes to the corresponding components. For example, each thing referred to in the building description needs its own thing description managed by the TDD and credentials which are provided to the Thing Credentials & Permission component.

#### 4.1.4. Intermediary API

The API for WoT intermediaries is defined by WoT, see (W3C (WoT Binding), 2020).

#### 4.1.5. Authentication

Recommended for most platforms. Currently, domOS recommend OAUTH2. See OAUTH2 specifications (OAuth Working Group).

### 4.2. Building Description

The initial components of the Building Description are Defined in Section 2.2.2. These components are subject to change and the final version of the Building Description will be released in the domOS deliverable D3.5, the final version of the domOS ontology. In the current version, the installed sensing and actuating capabilities are captured by placing domOS things in the defined building spaces. Spaces (specialized in rooms, floors, apartment, building) may be defined using the Building Description module of the ontology, in particular “`dco:space`”, and the static properties of the space. Things devices types are defined in the device hierarchy of the ontology (e.g. of types “`dco:supply-TemperatureSensor`”, “`dco:heatPumpAppliance`”). The relationship between space and device is given through the “`dco:hasDevice`” relation). Similar, a space (`dco:space`) may be exposed of a task through the “`dco:isExposedToTask`” relation. `dco:task` represents the goal for which a device or a set of devices are designed. For example, a heat pump and a heat pump relay are designed for the task of space heating.

Each device has a reference (“`dco:hasThingDescription`”) to the actual WoT thing description that implements the instance of the device type. This decouples the abstract, semantically well-defined, and stable measuring and control points defined by the ontology device type from the actual devices that supplies the data, thereby facilitating easier replacement and upgrade. The reference can be given as a relative URI containing an id (for lookup in the TDD), or an absolute URI to the TDD or external server where the TD is stored, or even a file (`file://device.txt`) on gateway hosted applications.

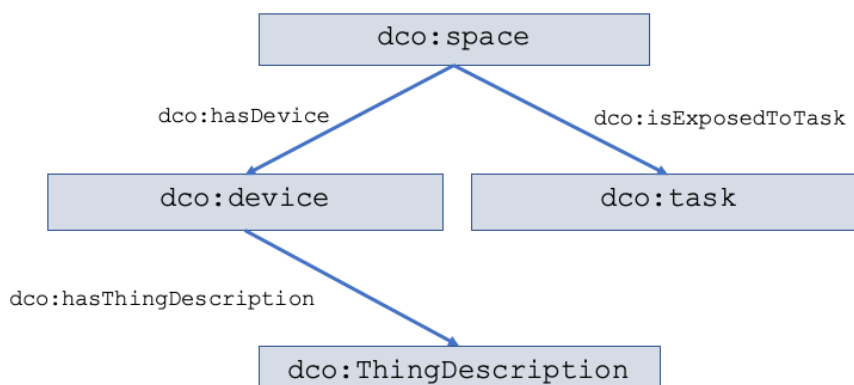


FIGURE 14: PRINCIPAL STRUCTURE OF THE BUILDING DESCRIPTION

In Figure 15, we present a Json-LD example of a Building Description. This building description has four main components: a specific building metadata (line 1 to 13), building topology (line 14 to 34) indicating the device location in each space, the building tasks (line 36 to 44), and the metadata of each device (line 46 to 74). A device can be a WoT Thing (e.g., waterTemperatureSensor01) or a device without a WoT Thing (e.g., heatPump01). We note that the final version of the BD will be released and specified by the final version of the domOS Common Ontology. A more detailed specification of the BD is presented at <https://www.dco.domos-project.eu/#desc>.

```

1 [{"@context": {"dco": "https://w3id.org/dco#"},
2  {"@id": "singleFamilyHouse01",
3   "@type": ["dco:SingleFamilyHouse"],
4   "dco:hasName": [{"@value": "Sion building"}],
5   "dco:hasEnvelopeType": {"dco:lightIsolation"},
6   "dco:hasThermalCapacity": [{"@value": "11234", "dco:unit": "dco
7   :megaWattHourPerSquareMeter"}],
8   "dco:hasConstructionYear": [{"@value": "1972"}],
9   "dco:hasCity": [{"@value": "Sion"}],
10  "dco:hasZipCode": [{"@value": "9220"}],
11  "dco:hasAddress": [{"@value": "55 Route de la liberte"}],
12  "dco:hasRenovationYear": [{"@value": "1989"}],
13  "dco:hasTotalSpace": [{"@value": "846"}],
14  "dco:hasFloor": [{"@id": "groundFloor01"}]
15  },
16  {"@id": "groundFloor01",
17   "@type": ["dco:groundFloor"],
18   "dco:hasSpace": [
19     {"@id": "Kitchen01",
20      "@type": ["dco:Kitchen"]},
21     {"@id": "ControlRoom01",
22      "@type": ["dco:ControlRoom"],
23      "dco:hasDevice": "heatPump01",
24      "dco:hasDevice": "waterBoiler01",
25      "dco:hasDevice": "smartMeter01",
26      "dco:hasDevice": "powerMeter01",
27      "dco:hasDevice": "gateway01",
28      "dco:hasDevice": "heatPumpRelay01",
29      "dco:hasDevice": "waterTemperatureSensor01"},
30     {"@id": "LivingRoom01",
31      "@type": ["dco:LivingRoom"],
32      "dco:hasDevice": "ambientTemperatureSensor01"},
33     {"@id": "Bedroom01",
34      "@type": ["dco:Bedroom"]}],
35   "dco:isExposedToTask": [{"@id": "spaceHeatingTask01"}, {"@id":
36   "waterHeatingTask01"}],
37   {"@id": "spaceHeatingTask01",
38    "@type": "dco:spaceHeatingTask",
39    "dco:isAccomplishedBy": [{"@id": "heatPump01"},
40    {"@id": "heatPumpRelay01"}, {"@id": "ambientTemperatureSensor01"},
41    {"@id": "powerMeter01"}]},
42   {"@id": "waterHeatingTask01",
43    "@type": "dco:waterHeatingTask",
44    "dco:isAccomplishedBy": [{"@id": "heatPump01"},
45    {"@id": "heatPumpRelay01"}, {"@id": "waterTemperatureSensor01"}]
46  },
47  {"@id": "heatPump01",
48   "@type": "dco:heatPump",
49   "dco:hasHeatPumpRelay": "heatPumpRelay01",
50   "dco:hasObservableProperty": {"HeatPumpPower01": {"@type": "dco
51   :powerProperty"}},
52   "dco:hasNominalPower": [{"@value": "3500", "dco:unit": "dco
53   :watt"}],
54   "dco:hasMaximumOffTime": [{"@value": "0.5", "dco:unit": "dco
55   :hour"}],
56   "dco:hasMinimumOnTime": [{"@value": "8", "dco:unit": "dco
57   :hour"}]},
58  {"@id": "waterBoiler01",
59   "@type": "dco:waterBoiler",
60   "dco:hasObservableProperty": {"waterTemperature01": {"@type": "dco
61   :waterTemperatureProperty"}},
62   "dco:hasVolume": [{"@value": "0.8", "dco:unit": "dco
63   :squareMeter"}]},
64  {"@id": "waterTemperatureSensor01",
65   "@type": "dco:waterTemperatureSensor",
66   "dco:hasPropertyAffordance":
67   {"waterTemperaturePropertyAffordance01": {"@type": "dco
68   :waterTemperaturePropertyAffordance", "dco:unit": "dco:Celsius",
69   "dco:hasFeatureOfInterest": "boiler water temperature"}},
70   "dco:observes": {"waterTemperature01": {"@type": "dco
71   :waterTemperatureProperty"}},
72   "dco:hasThingDescription": "https://domos.oiken.ch/tdd
73   /waterTemperatureSensorID"},
74  {"@id": "powerMeter01",
75   "@type": "dco:powerMeter",
76   "dco:hasPropertyAffordance": {"power01": {"@type": "dco
77   :powerPropertyAffordance", "dco:unit": "dco:watt", "dco
78   :hasFeatureOfInterest": "heat pump power"}},
79   "dco:observes": {"HeatPumpPower01": {"@type": "dco:powerProperty"}},
80   "dco:hasThingDescription": "https://domos.oiken.ch/tdd
81   /powerMeterID"},
82  {"@id": "ambientSensor01",
83   "@type": "dco:ambientSensor",
84   "dco:hasPropertyAffordance": {"temperatureProperty01": {"@type":
85   "dco:temperaturePropertyAffordance", "dco:unit": "dco:Celsius",
86   "dco:hasFeatureOfInterest": "room temperature"}},
87   "dco:hasPropertyAffordance": {"humidityProperty01": {"@type": "dco
88   :humidityPropertyAffordance", "dco:unit": "dco:Percentage", "dco
89   :hasFeatureOfInterest": "room humidity"}},
90   "dco:hasEventAffordance": {"temperatureOverheatingEvent01":
91   {"@type": "dco:temperatureOverheatingEventAffordance", "dco
92   :unit": "dco:Celsius"}},
93   "dco:hasThingDescription": "https://domos.oiken.ch/tdd
94   /ambientSensorID"}
95  ]

```

FIGURE 15: EXAMPLE OF A BUILDING DESCRIPTION IN JSON-LD

## 5. Conclusion and Future Work

### 5.1. Conclusion

This document has advanced the high-level requirements into a functional specification of the domOS IoT eco-System. The functional specification consists of a simple 4-layered reference model for the eco-system, a specification of required, recommended, and optional features. The main provided and consumed interfaces (APIs) among the core components have been identified. Guidance is given for the main components of the platform and their functional responsibilities. The dynamic interactions among the components are illustrated through sequence diagrams that in detail exemplifies how the system is intended to behave to solve key tasks. Finally, the details of key APIs are defined as REST APIs using the state-of-the-art tool swagger.

In conclusion, the document has presented the domOS functional specification in sufficient detail to allow prototype implementation of the platform instances. It is also clear that this implementation work will help identifying many of the details that would be part of a detailed and exhaustive technical specification.

### 5.2. Future work

Obviously, the key future work is to prototype the eco-system, a necessary step to ensure that the specification is practically implementable, and usable for the demonstrator scenarios.

It is left for future work to define a fully **self-serve platform** that supports self-registration of Building Owners, Service providers, and Applications. A further step could be to multiple platform instances and thereby platform operators share a common registry of applications. This would ease service providers to distribute their applications once domOS become widespread in use.

The current functionality does not specify the **caching-behaviour** of the intermediaries, but it is conceivable that an intermediary caches recently read values, and stores “last-known” values of properties. Similarly, an extension could include the digital-twin behaviour laid out in the WoT specifications. It is also conceivable that the Building Description Directory can be served as a “Building Thing” using WoT-technology.

The current domOS common **ontology could be** extended to capture dependencies and influences between devices (more generally between the different observation and control points in the building).

The **virtualization layer** provided by the building model could be revised, such that it is the building description’s responsibility to describe the possible properties that can be access in the building. Currently, these are mainly characterized by the device types that exists in a building, and by the type of annotation of the thing’s affordances. The main advantage is a further decoupling of the between applications and the underlying devices, freeing applications from dealing with several different devices and “things” specific. However, such an extended virtualization layer must be carefully designed, and semantically well-supported by the final version of domOS ontology, which will be available in Deliverable D3.5.

## 6. References

**CEN-CENELEC-ETSI Smart Grid Coordination Group** Smart Grid Reference Architecture [Online]. - 11 2012. -

[https://ec.europa.eu/energy/sites/ener/files/documents/xpert\\_group1\\_reference\\_architecture.pdf](https://ec.europa.eu/energy/sites/ener/files/documents/xpert_group1_reference_architecture.pdf).

**domOS D2.1** D2.1 Report on Requirement Analysis for IoT Ecosystem [Online] // <https://www.domos-project.eu/>. - 11 2, 2021. - <https://www.domos-project.eu/filedelivery.php?docId=20>.

**domOS D3.2** D3.2 domOS Common Ontology\_Initial Version [Online] // <https://www.domos-project.eu/>. - 11 02, 2021. - <https://www.domos-project.eu/filedelivery.php?docId=17>.

**Le Guilly T. [et al.]** Model Checking Feature Interactions [Conference] // Communications in Computer and Information Science. - 2016. - pp. 307-325.

**Le Guilly T.** Model Based Analysis of Embedded Software for Smart Homes [Report]. - 2016.

**OAuth Working Group** OAuth Working Group Specifications [Online] // OAuth. - <https://oauth.net/specs/>.

**Pedersen T.** Smart Home Models - Analysis, Simulation and Synthesis [Report]. - 2018.

**SmartBear Software** OpenAPI Specification [Online] // Swagger. - <https://swagger.io/resources/open-api/>.

**The Internet Society** A Universally Unique Identifier (UUID) URN Namespace [Online] // Internet Engineering Task Force. - 2005. - <https://datatracker.ietf.org/doc/html/rfc4122>.

**W3C (WoT Architecture 1.1)** Web of Things (WoT) Architecture 1.1 [Online] // <https://www.w3.org/>. - 11 2020. - <https://www.w3.org/TR/2020/WD-wot-architecture11-20201124/>.

**W3C (WoT Architecture)** Web of Things (WoT) Architecture [Online] // <https://www.w3.org/>. - 4 2, 2020.

**W3C (WoT Binding)** Web of Things (WoT) Binding Templates [Online] // <https://www.w3.org/>. - 1 30, 2020. - <https://www.w3.org/TR/wot-binding-templates/>.

**W3C (WoT Discovery)** Web of Things (WoT) Discovery [Online] // <https://www.w3.org/>. - 6 2, 2021. - <https://www.w3.org/TR/wot-discovery/>.

**W3C (WoT Scripting API)** Web of Things (WoT) Scripting API [Online] // <https://www.w3.org/>. - 11 24, 2020. - <https://www.w3.org/TR/wot-scripting-api/>.

**W3C (WoT TD)** Web of Things (WoT) Thing Description [Online] // <https://www.w3.org/>. - 4 9, 2020. - <https://www.w3.org/TR/wot-thing-description/>.

**Wikipedia** MoSCoW method [Online] // <https://en.wikipedia.org/>. - [https://en.wikipedia.org/wiki/MoSCoW\\_method](https://en.wikipedia.org/wiki/MoSCoW_method).



## Appendix A: Requirements Review and Consolidation

This section reviews, consolidates, and prioritizes the high-level requirements from domOS D2.3. The requirements are reproduced in Section “**Error! Reference source not found.**” where they are annotated with labels to enable traceability. Table 3 explains the notation for traceability labels.

The requirements are prioritized using the MoSCoW principles (Wikipedia):

1. Must-have (Highest)
2. Should-have
3. Could-have
4. Wont-have (this time). (Lowest)

Requirement’s priority is given such that a minimum viable domOS can be achieved by implementing all Must-Have requirements.

Tags are used to give meaning full name to requirement and also label the source (for traceability) of the detailed technical requirement from the high-level specification in D2.1.

**TABLE 3: REQUIREMENTS REVIEW**

Tags	Comment	Responsible Component	Priority
<b>Multiplicity</b> I1 I2 I6 I7	<p>One DOPL instance must support multiple containers (hence multiple costumers). A container can only be part of one DOPL instance.</p> <p>One DOPL instance may serve multiple applications. An application may use multiple DOPL instance.</p> <p>One costumer manages exactly one container. A container belongs to exactly 1 costumer (accepted as a design restriction in a first version: low priority)</p> <p>One container registers multiple smart-systems, but a smart system only belongs to 1 container.</p>	DOPL	Must
<b>Multiplicity (derived)</b>	DOPL must keep its containers securely isolated (multi-tenacy).	DOPL	Must
<b>WoT Standard</b> EF1-EF4	<p>DOPL must use and adhere to the WoT specifications.</p> <p>A smart system is represented in DOPL by 1 or more “Things” or “Intermediaries”,</p> <p>A smart system presents its “Things Description” and is registered in a “Thing directory” per container.</p> <p>The registration is done by Costumer</p> <p>An application is represented by 1 or more “WoT Consumers”</p>	DOPL	Must
<b>Smart Systems (Things) Directory</b> ES12 ES14	DOPL hosts a WoT TD per container and properly isolated	WoT TD (per container)	Must
<b>Mediator functionality</b> P1, P6	<p>DOPL forwards messages from application to correct smart system device,</p> <p>DOPL performs protocol conversion from DOPL unified protocol to device specific protocol</p> <p>The unified DOPL protocol is based on REST principles and SSE (+server-side events, or similar mechanism).</p> <p>Perform data-conversion to/from domOS ontology and device parameters.</p>	WoT Intermediaries	Must

<p><b>No direct links</b> P1, P5, SR4</p>	<p>“An application cannot access the smart system without authentication and authorization from DOPL.</p> <p>Any communication between Application and smart system takes place via DOPL.</p> <p>However, requirement A3 states that “from business service perspective, applications interact directly with smart systems”. We interpret this to mean that the intermediary should be transparent at least for the business process, but preferably also the application implementation.</p>	<p>WoT Intermediaries</p>	<p>Must</p>
<p><b>Compulsory mediator:</b> ES8, ES11</p>	<p>DOPL: any application using domOS may only use devices registered in DOPL, and accesses these via the DOPL Intermediary.</p> <p>Problem: is it realistic that all devices are domOS?</p> <p>One partner has required the possibility of having direct access to the “thing”. DOPL should allow for this, but the building owner should explicitly consent. This also enables that the platform (and security checks per device access) can be bypassed avoiding it becoming a bottleneck in cases of intense communication.</p>	<p>WoT Intermediaries</p>	<p>Should</p>
<p><b>Applications</b> ES4</p>	<p>(Platform operator) registers possible applications/services</p> <p>Customer activates/de-activates applications, and subscriptions.</p>	<p>WoT Consumer  Application Manager</p>	<p>Should</p>
<p><b>Privacy-rules.</b> P9, SP1, SP6 SP5, SP7</p>	<p>What is a privacy rule? Need a notation to describe privacy rules. Need a UI for describing these, need a mechanism registering and checking and enforcing the rules.</p>	<p>Security and Privacy Manager</p>	<p>Wont</p>
<p><b>Privacy-mechanism</b> P9, SP1, SP6 SP5 SP7</p>	<p>DOPL must enable Customers to define access rights (e.g., RWX) for each Thing’s affordances. We remark that this is a very low-level mechanism.</p> <p>DOPL must enable customers with the ability to configure, change (add, revoke) access rights, i.e., they may be subject to leasing.</p> <p>DOPL must provide a comprehensible digest about which applications have what kind of access to what smart system attributes.</p>	<p>Security and Privacy Manager</p>	<p>Must</p>



<b>GDPR</b> <b>SP2, SP4</b>	<p>When DOPL need to store data, it must do so in a GDPR compliant manner.</p> <p>Examples of data that the platform may need to store are e.g., Costumer accounts (credentials, email-address, phone numbers for notifications) access logs, device temporary data)</p> <p>A DOPL is owned by the platform operator and data is owned by the end-user (building owner).</p> <p>Remark SP2 is incomplete, and a contradiction to SP4.</p>	DOPL	Must
<b>Secure connection</b> <b>P7 P8 A4</b>	<p>Communication is encrypted, authenticated, and authorized between</p> <ul style="list-style-type: none"> <li>• App and DOPL,</li> <li>• DOPL and smart system, and</li> <li>• (APP and smart system)</li> </ul>	Authentication and Authorization, AIM, and all external communication channels	Must
<b>Smart systems descriptions</b> <b>OS2, ES3</b> <b>ES1, ES2</b>	<p>DOPL must have a functionality for registering participating smart systems and “descriptions” thereof and making ontology annotations.</p> <p>DOPL must support adding domOS ontology annotations</p>	Extended Wot Things descriptions	Must
<b>Smart systems management</b>	DOPL support for lifecycle management of smart systems (commission, update/ maintain (e.g., battery change)), activate/deactivate, de-commission)	Things Manager and UI	Could
<b>OS1?</b>	DOPL must expose building concepts of the enclosing container as defined in the ontology, e.g., structure and layout of building and where which smart systems are installed.	Building description	Should
<b>Smart systems connection</b>	The platform must support smart systems implemented via gateways/edge devices, cloud, and devices that are natively “smart” enabled (WoT Compliant devices).	Things / Intermediate abstractions	Could
<b>D2</b> <b>EF9</b>	DOPL must be designed to support flexible deployment options such that it can be deployed to both a gateway/edge at costumer premises, outside costumer premises, cloud, or a combination thereof (e.g., distributed deployment where some DOPL components are cloud hosted and some are GW/Edge hosted?)	DOPL	Should

	<p>Scalable design (stateless modules etc.)</p> <p>This means that DOPL architecture design should use a loosely coupled component-based design approach.</p>		
<p>“One application to set a control point”. P10 ES6</p>	<p>To enforce this, DOPL must know what applications want to write to what things/things affordances/actuators.</p> <p>See discussion in note 1).</p>	<p>Application Manager Note 2)</p>	<p>Could</p>
<p>“Check of required function” A2 ES6</p>	<p>A2 states that the <b>application</b> is responsible for executing a compliance check before activation.</p> <p>The application should be able to query DOPL semantically for smart system capabilities to determine whether it can be activated.</p>	<p>Application and Building model</p>	<p>Could</p>
<p>Smart system credentials SR4 SR2</p>	<p>DOPL must provide a mechanism for storing, adding, removing, updating smart systems credentials in various formats as needed by the underlying smart systems.</p> <p>These credentials must only be used internally in DOPL.</p>	<p>Authorization and Authentication</p>	<p>Should</p>
<p>Application credentials SR3 SR4</p>	<p>“Secure connection”: Two-way authorization: the smart systems must authorize the application, and the application must authorize the smart system (via DOPL) (be ensured that it uses the right smart systems only).</p> <p>The platform must make it possible for Costumers to maintain application credentials (create, change, revoke, destroy).</p> <p>Authentication may be subject to leasing to enable revocation</p>	<p>A&amp;A  Credentials Manager</p>	<p>Must</p>
<p>Platform security</p>	<p>All communication: between platform internal components (when distributed), must be secured.</p>	<p>DOPL</p>	<p>Must</p>
<p>EF4, EF6 Fault-tolerant and safe operation</p>	<p>The system must support fault tolerant operation, i.e., the ability to continue (possibly degraded) service in case of failure in the DOPL, applications, or smart system. DOPL must enable local fallback-modes, detect and report failures to the parties that can handle them, ultimately the</p>	<p>Health-monitor &amp; notification service</p>	<p>Could</p>

	<p>customer. Hence, monitoring application and smart-system, and platform health is desired.</p> <p>(Safety means that no harm or injury happens to the costumers, inhabitants, etc.).</p>		
<b>Customer user interfaces</b> <b>EF7 EF8</b>	<p>Management UIs must exist for administrating (adding, removing, changing) costumers, smart systems registration plus ontology annotations, application subscriptions, and credentials, permissions.</p> <p>All end-user facing administration interfaces must be understandable and usable by ordinary persons. This implies a professional UX design and usability evaluation as part of the development.</p>	User Mgt UI SS Mgr UI Subscription Mgt UI Credential Mgt UI Privacy Mgt UI	Should
<b>EF9</b>	Single point of failure		
<b>Feature interaction</b> <b>EF6? P10</b>	Undesired feature interaction is not to be handled by DOPL		Wont

## Note 1)

Essentially this means that the platform must be able to grant an application exclusive access to a control point. The platform must track which control points are allocated to what services. If an application can allocate control-points **dynamically**, the system has potentials for deadlocks. Solutions may be to require applications to allocate all control-points in advance, allocate control-points in a globally order decided by platform, or the platform should detect and recover from deadlocks (typically by forcefully terminating applications). In general, exclusive access to control-points is an insufficient condition to avoid undesired feature interaction among applications.

Business services / applications provide a manifest of needed properties that the platform can check?

Furthermore, tracking the state of locks is inherently stateful, requiring special recovery mechanisms from failures. Hence, a **static map** from applications to needed control-points is easier to handle (but less flexible). When applications are long running implementing continuous services, this may less of a problem, whereas if applications are one-shot executions, it may be too inflexible.

A WoT Thing provides 4 kinds of “affordances”: Properties, Actions, Events, and Navigation Links. A property can be read/written, and an action may change or only read. Hence, a thing may have multiple write-entries to the same underlying physical control “relay(s)” (e.g., property ON/OFF, and action “turnON”). How should a “control-point” map to the underlying mechanism that can only be controlled via one application: 1) entire “thing” is allocated to each application, or 2) individual affordances.

-> add annotations?

P11: “Interferences must be handled at service level”: Unclear requirement: Does it mean that a service should know which services it is compatible with, and not be installable if such a service is already running? A given service may not now in advance what other services may be running. That means that the platform may reveal what other services are running?

- Not trivial, needs lot of knowledge of what the service wants to do and the general profession rules
- Possibility: lock/unlock, but risk of deadlocks and does not solve cross-system problems

Note 2)

*“Service orchestration supports the integration of multiple services to perform a user task or data synchronization in real time [57]. In the IoT context, orchestration is concerned with the identification of which components or smart devices are needed to form the requested service [11]. An orchestrator can be any IoT device that is used to control the execution transparently to the user. The orchestrator sends a triggering event that checks the condition for carrying out an action using actuators [58]. The development of a service orchestrator requires a deep understanding of service semantics and decomposition of the service request [59]. “*

## Appendix B: Requirements Links and Traceability

The following section adds requirement labels for the requirements for the domOS eco-system listed in D2.1 (domOS D2.1, 2021). The section repeats the requirements and adds labels to the individual requirements and thereby enabling traceability between design decisions and the original high-level requirements.

The labelling used are as follows:

TABLE 4: REQUIREMENTS TRACEABILITY LABELS

Label number	Requirement Category
<b>On</b>	Objectives and Success Criteria
<b>In</b>	IoT Eco-system
<b>Pn</b>	Platform requirements
<b>An</b>	Application requirements
<b>Sn</b>	Smart Systems requirements
<b>SRn</b>	Security requirement
<b>OSn</b>	Ontology and Semantics requirement
<b>SPn</b>	Support for Privacy
<b>Dn</b>	Deployment requirement
<b>ESn</b>	Smart systems description
<b>EFn</b>	Extra-functional requirements

## 7. Objectives and Success Criteria

The WP2 objectives and their corresponding success criteria are presented in Table 5.

TABLE 5: SUCCESS CRITERIA

Objective	Success criteria
<p>To elaborate and prototype a standard-based architecture named “IoT ecosystem”, allowing:</p> <ul style="list-style-type: none"> <li>a decoupling between smart systems and applications,</li> </ul>	<ol style="list-style-type: none"> <li>An application can interact with any container, providing smart systems make the appropriate monitoring and control parameters available. <b>O1</b></li> <li>Smart systems provide a description of: <ul style="list-style-type: none"> <li>their monitoring and control parameters using the domOS core ontology, <b>O2</b></li> <li>the access protocol(s) along with the security credentials, and of <b>O3</b></li> <li>the syntax of exchanged messages. <b>O4</b></li> </ul> </li> <li>An application can verify whether the container is equipped with the required monitoring and control parameters before deployment. <b>O5</b></li> <li>The IoT ecosystem is based on open standards promoted by recognised standardisation bodies. <b>O6</b></li> </ol>
<ul style="list-style-type: none"> <li>owners to manage their privacy, and</li> </ul>	<ol style="list-style-type: none"> <li>Owners explicitly allow an application to operate with their smart systems. <b>O7</b></li> <li>Owners explicitly allow applications to monitor/ control individual parameters. <b>O8</b></li> <li>Platforms provide owners with an interface where they can centrally manage their privacy. <b>O9</b></li> </ol>
<ul style="list-style-type: none"> <li>secure operation of buildings.</li> </ul>	<ol style="list-style-type: none"> <li>Applications dispose of their own security credentials to access the platform. <b>O10</b></li> <li>The platform disposes of the security credentials to access smart systems. <b>O11</b></li> </ol>
<p>To upgrade the three participating IoT platforms (S-IOT, cloud.iO, ArrowHead) according to the defined IoT ecosystem.</p>	<ol style="list-style-type: none"> <li>Each participating platform provides an implementation of the IoT ecosystem. <b>O12</b></li> </ol>
<p>To prototype a smart application capable of running over the three enhanced IoT platforms.</p>	<ol style="list-style-type: none"> <li>A Proof of Concept (PoC) illustrates that a single prototype application is capable to operate over each platform. Each platform connects smart systems providing the same monitoring and control functions (e.g., electrical power monitoring and power supply control). <b>O13</b></li> </ol>
<p>To define a sound concept for retrofitting existing buildings with monitoring and control infrastructure for energy appliances.</p>	<ol style="list-style-type: none"> <li>The concept allows to retrofit all generations of buildings throughout Europe. <b>O14</b></li> <li>Monitored and controlled parameters enable smart services for energy efficiency and energy flexibility. <b>O15</b></li> <li>The overall cost (hardware, manpower) for installation is in-line with the expected benefits. Target values: 100 € for a communication gateway, 1 hour work time onsite and 100 € hardware cost per connected appliance. <b>O16</b></li> </ol>



## 7.1. Overview

### 7.1.1. Overview of the IoT Ecosystem

For simplicity, a customer is assumed to manage exactly one container. **I1**

The central element of a domOS ecosystem is the platform. Once they have committed to a platform, customers can register one or more smart systems on it. **I2**

Customers can subscribe to applications. **I3** Applications can only be activated if the smart systems in the corresponding containers provide appropriate monitoring and control parameters, **I4** and if it is compliant with privacy rules defined by customers. **I5**

An overview of the IoT ecosystem is provided on Figure 16.

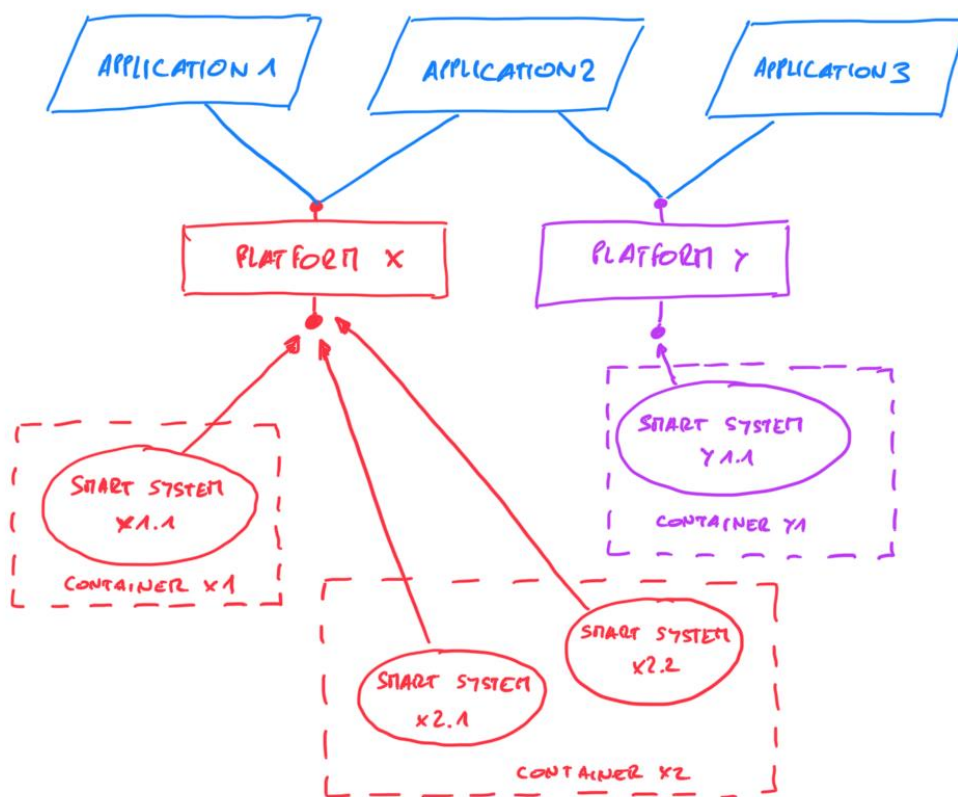


FIGURE 16: ELEMENTS IN THE IOT ECOSYSTEM

Compliant platforms implementing the IoT ecosystem can be many, possibly based on different implementations. **I6** A container is hooked to only one platform. On the contrary, applications can access multiple platforms. **I7**



## 7.2. Functional Requirements

### 7.2.1. Requirements for Platforms

A platform is a compulsory mediation point between applications and smart systems, i.e., there are no direct links between applications and smart systems. **P1**

A platform is a pure ICT player. As such, it does not care about the business aspects of smart services. **P2**

A platform shall:

- have access to a description of the smart systems hosted by participating containers, **P3**
- allow a registered application to verify whether a container disposes of the appropriate infrastructure for the service, **P4**
- act as an intermediary between applications and smart systems. **P5**

In its intermediary function, the platform shall:

1. handover messages from applications to smart systems, and from smart systems to applications, **P6**
2. accept secured connection from registered applications, **P7**
3. implement secure connections to registered smart systems, and **P8**
4. implements privacy rules defined by the customers. **P9**

At most one application shall control a set point in a smart system. This rule shall be enforced by a platform on a “first come – first serve” basis. **P10** Interferences between multiple services accessing multiple actuators (e.g., one Service turns on the heater and one other Service opens a window) shall be managed at the Service level. **P11**

### 7.2.2. Requirement for Applications

An application is a software component that provides a service through interaction with smart systems. **A1**

Before activation, an application shall verify that smart systems inside a container offer the required functions. **A2**

From a service perspective, applications interact directly with smart systems. **A3**

From a technical perspective, applications connect to a platform. An application shall use a unique set of security credentials defined by the platform, i.e., it does not share any security credentials with smart systems. **A4**

### 7.2.3. Requirement for Smart Systems

Containers host one or more connected system(s) featuring internet connectivity. **S1**

Smart systems are connected systems made compliant with the domOS ecosystem. **S2** Turning a connected system into a smart system should only require the provision of a description of the connected





system (i.e., no protocol adaptation, no message translation, no modified security scheme or security credentials). **S3**

Considering that a platform is the peer communication entity of a smart system, and that platforms can support a limited set of protocols, the integration of systems only through description is only possible if the connected systems implement protocols, message formats and access control schemes supported by platforms. **S4** The legacy technologies supported by platforms shall be defined. **S5** In this document, only systems whose communication is supported by platforms are considered. Other systems require an application-level gateway. **S6**

A smart system shall register on a platform. As part of the registration process, it makes its description available to the platform. **S7**

The component turning a connected system into a smart system can be implemented either on a local gateway in the container premises, as a cloud service, or natively in the connected system. **S8**

A smart system description can be either static (i.e., typically provided in a text file), if the smart system configuration remains stable over time, or dynamic (i.e., generated from a smart system internal directory). **S9**

#### 7.2.4. Security Requirements

Smart systems are assumed to feature a secure (i.e., authenticated and encrypted) data interface. **SR1**

Security credentials for smart systems shall be uploaded in the platform, e.g., as part of their description. **SR2**

Applications establish a secure connection to platforms, using a (state-of-the-art) security scheme and a security credential provided by platforms. **SR3**

Applications shall in no circumstances know smart systems security credentials. **SR4**

#### 7.2.5. Requirements on Semantics

WP3 “Common Ontology and Semantics” develops the domOS core ontology, which will define naming conventions for relevant concepts in buildings. **OS1**

Smart systems descriptions should associate domOS core ontology elements with information enabling the concrete remote (read and/or write) access to current element values in the smart system. **OS2**

#### 7.2.6. Requirements on Privacy

A platform shall be able to grant or deny applications the right to access single monitoring or control points<sup>2</sup>. **SP1**

Ensuring a legitimate use of the collected data (e.g., purpose limitation and data minimization according to GDPR) is the sole responsibility of the application. **SP2**

---

<sup>2</sup> In the domOS vision, the access rights for applications should be managed by customers. The implementation of such a privacy management system is not included in the requirements.

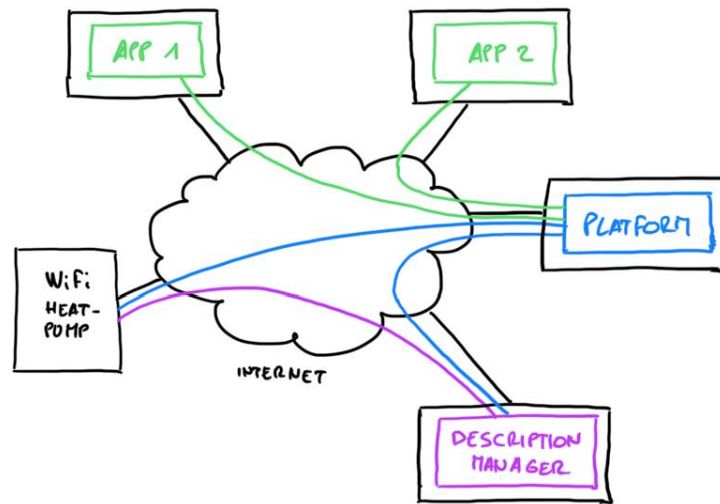


### 7.2.7. Requirements on Deployment Topology

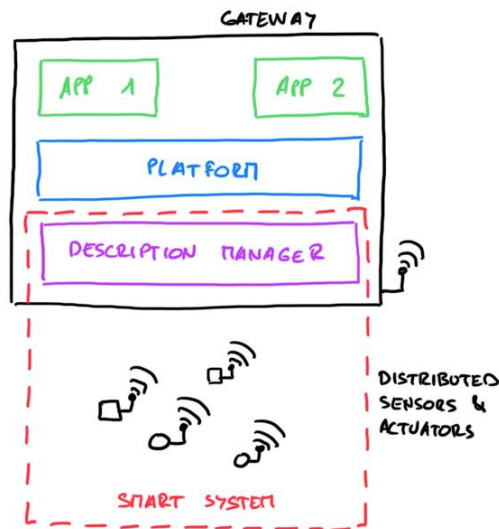
Connected systems are physical devices / appliances installed in the customer premises. **D1**

A platform and an application are components that can be deployed using different topologies: **D2**

It should be possible to use edge and cloud hosting at all levels. Two possible topologies are presented on Figure 17.



(a)



(b)

FIGURE 17: TWO POSSIBLE TOPOLOGIES: (A) DISTRIBUTED CLOUD SOLUTION (B) EDGE SOLUTION



The choice of a topology appropriate for a given context is outside the scope of the WP2. **D3**

The IoT ecosystem should put as few constraints as possible on underlying topologies. **D4**

IoT platforms may not support all possible topologies. **D5**

### 7.2.8. Support of Privacy

Compliance with data protection regulations must be insured by applications. **SP3**

Platforms play a technical mediation role and do not store any personal data. Hence, they are not subjected to data protection regulations **SP4**. However, they support customers for privacy management **SP5**. As part of the subscription process to an application, a platform should give access to monitoring or control parameters only after a formal approval by the customer **SP6**.

Platforms are also able to present to customers a report on implemented access rights (which application may access which monitoring / control parameters). **SP7**

## 7.3. Non-Functional Requirements

### 7.3.1. Compliance with Recognized IoT / Web Standards

“The Internet of Things (IoT) is widely recognised to have lots of potential, but its commercial potential is being held back by fragmentation. A sensor on its own has limited value, but there are huge opportunities for open markets of services that combine sensors, actuators and multiple sources of information. The Web of Things seeks to counter the fragmentation of the IoT, making it much easier to create applications without the need to master the disparate variety of IoT technologies and standards. Digital twins for sensors, actuators and information services are exposed to consuming applications as local software objects with properties, actions and events, independently of the physical location of devices or the protocols used to access them.”<sup>3</sup>

The Web of Things Interest Group at W3C leads standardization for the so-called Web of Things (WoT). W3C WoT standards have been identified as the most relevant standard series for the smart buildings use cases (see extract from the WoT architecture presented above).

Relevant WoT standards with their current status are reported on Table 6:

TABLE 6: STATUS OF W3C WOT STANDARDS

Title	Date for current version	URL for the latest published version	Content description	Status
Web of Things (WoT): Use Cases and Requirements	27 January 2021	<a href="https://w3c.github.io/wot-usecases/">https://w3c.github.io/wot-usecases/</a>	Collection of new IoT use cases from various domains	Draft

<sup>3</sup> From “Web of Things (WoT) Architecture - W3C Recommendation 9 April 2020” available at <https://www.w3.org/WoT/>



Title	Date for current version	URL for the latest published version	Content description	Status
Web of Things (WoT) Architecture	9 April 2020	<a href="https://www.w3.org/TR/wot-architecture/">https://www.w3.org/TR/wot-architecture/</a>	Description of the abstract architecture for the W3C Web of Thing	Recommendation
Web of Things (WoT) Thing Description	9 April 2020 (corrected 23 June 2020)	<a href="https://www.w3.org/TR/wot-thing-description/">https://www.w3.org/TR/wot-thing-description/</a>	Formal model and a common representation for a Web of Things (WoT) Thing Description	Recommendation
Web of Things (WoT) Binding Templates	30 January 2020	<a href="https://www.w3.org/TR/wot-binding-templates/">https://www.w3.org/TR/wot-binding-templates/</a>	Binding Templates enable a Thing Description to be adapted to the specific protocol or data payload usage across the different standards.	Working Group Note (draft)
Web of Things (WoT) Scripting API	24 November 2020	<a href="https://www.w3.org/TR/wot-scripting-api/">https://www.w3.org/TR/wot-scripting-api/</a>	Application programming interface (API) representing the WoT Interface that allows scripts to discover, operate Things and to expose locally defined Things.	Working Group Note (draft)
Web of Things (WoT) Security and Privacy Guidelines	6 November 2019	<a href="https://www.w3.org/TR/wot-security/">https://www.w3.org/TR/wot-security/</a>	Guidance on Web of Things (WoT) security and privacy.	Working Group Note (draft)

The IoT ecosystem for buildings shall specialize the WoT Architecture for the specific context of smart buildings. **EF1**

Smart system description shall be compliant with the WoT Thing Description (TD). Links to smart systems on the field should follow the WoT Binding Templates guidelines. **EF2**

The WoT Scripting API is described in a working document and features a reference implementation<sup>4</sup>. It could be a valuable tool to upgrade the platforms but is not a compulsory element of the IoT ecosystem for buildings. **EF3**

The WoT Security and Privacy Guidelines describes threats scenarios and proposes recommendations based on the best available practices in the industry. As it contains informative statements only, it is not part of the IoT ecosystem. **EF4**

### 7.3.2. Safety

Smart systems shall work safely even if the connection to the platform or to an application is lost. If a smart system does not meet this requirement, it should not be part of the ecosystem. Hence, network or platform failures may decrease the performance of the subscribed smart systems but shall not jeopardize their safety. **EF4**

<sup>4</sup> <http://www.thingweb.io/>



The platform operators decline any liability regarding the safety of connected smart systems. **EF5**

Application could cause safety problems by providing erroneous set points (e.g., frequent switch on / switch off commands on a heat pump). The liability for such safety issues shall be handled directly by customers and application operators. **EF6**

### 7.3.3. Usability

Management procedures for registering customers, adding / removing smart systems, and subscribing / unsubscribing to applications shall be simple enough to be executed by ordinary persons. **EF7**

The usability of management procedures depends on the concrete implementation of a platform but also on features of the IoT ecosystem. The IoT ecosystem should enable the development of compliant platforms featuring a high degree of usability. **EF8**

### 7.3.4. Performance

The IoT ecosystem should allow scalable implementations. **EF9** This means, but is not limited to:

- using lightweight protocols,
- enabling implementation in independent stateless modules,
- minimizing the volume of stored data,
- allowing parallel implementation in clusters.

## 7.4. Smart System Description

The IoT ecosystem shall provide guidelines on the schema of the smart system description. These guidelines shall be based on the Things Description specification defined by the WoT architecture. **ES1**

The smart system description shall use of the domOS core ontology, to name monitoring and control parameters in participating smart systems. **ES2**

Platforms should have access to the description of registered smart systems. **ES3**

## 7.5. Subscription to an Application

The following preconditions are assumed to be fulfilled: Customer C is registered on the platform P, application A are registered on the platform P, smart systems SS1 and SS2 have registered their description on platform. **ES4**

In this context:

- Customer C initiates a subscription process to Application A. **ES5**
- Platform P verifies that the smart systems SS1 and SS2 comply with A's requirements. If not, the subscription process is aborted. The verification process makes use of the Directory scenario presented in Section 2.2.7. **ES6**
- Platform P enables A to access the required parameters after approval of the customer C. The application A is from now on active. **ES7**



## 7.6. Intermediary Function

The intermediary function as introduced in Section Requirements for Platforms 7.2.1 should be based on the Intermediary concept defined in the WoT architecture<sup>5</sup>. **ES8** This concept is illustrated in Figure 18.

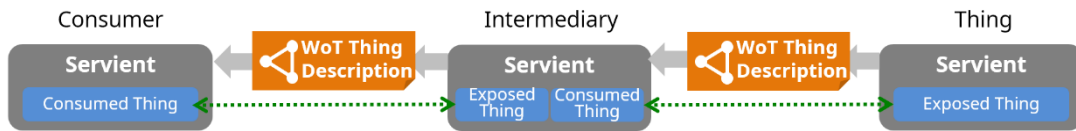


FIGURE 18: ILLUSTRATION OF THE INTERMEDIARY CONCEPT IN THE WOT ARCHITECTURE (W3C (WOT ARCHITECTURE), 2020)

The intermediary acts as a proxy between Applications and Things. **ES9**

The implementation of the intermediary should be based on the servient concept defined in [WoT]. **ES10** As illustrated on Figure 19, Smart Systems “expose” their descriptions (“Thing Description”) to the servient that “consume” them. The servient merges the descriptions into a new description, “exposed” to Applications.

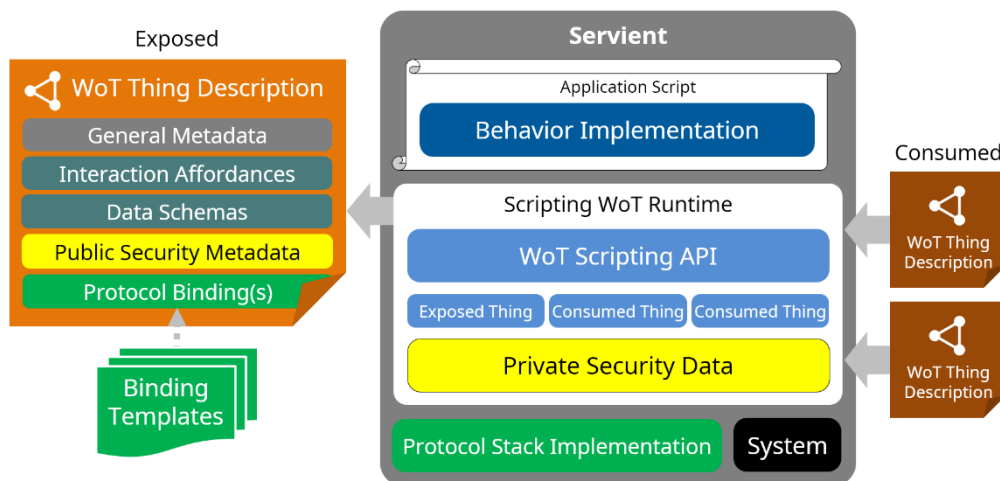


FIGURE 19: IMPLEMENTATION OF THE INTERMEDIARY BASED ON THE SERVIENT CONCEPT (W3C (WOT ARCHITECTURE), 2020)

The servient implementation should make use of the WoT Scripting API (W3C (WoT Scripting API), 2020), which allows an abstract<sup>6</sup> and semantic access to Smart Systems (Things). Hence, the communication

<sup>5</sup> “An entity between Consumers (Applications in the present document) and Things (Smart systems) that can proxy, augment, or compose Things and republish a WoT Thing Description that points to the WoT Interface on the Intermediary instead of the original Thing. For Consumers, an Intermediary may be indistinguishable from a Thing, following the Layered System constraint of REST.” Web of Things (WoT) Architecture. W3C Recommendation 9 April 2020

<sup>6</sup> In this context, “abstract” means “independent of communication protocols and message formats”.



(protocols and message formats) between an Application and the Platform could differ from the communication between the Platform and a Thing.

The intermediary should implement the access right mechanism defined in 7.2.4. **ES11**

## 7.7. Smart Systems Directory

The scenario “Subscription to an Application” presented in Section 7.5 requires that the Platform manages a Smart System directory. **ES12**

The description of smart systems should be uploaded in the directory. During the subscription process, applications should express their requirements in such a way that the Platform can check whether the requirements are fulfilled by asking the directory. **ES13**

The WoT Architecture defines a “Thing Directory” role and refers to the IETF draft standard (CoRE Resource Directory, <https://tools.ietf.org/html/draft-ietf-core-resource-directory-21>) **ES14**

