Highlander

High performance computing
to support smart land services

D4.4 Data Harmonization Specifications

Deliverable Lead	DIBAF, Dedagroup
Deliverable due date	2021/03/31
Status	DRAFT
Version	V1.0

Co-financed by the Connecting Europe
Facility of the European Union

Highlander
High performance computing
to support smart land services

DOCUMENT CONTROL PAGE

Title	D4.4
Creator	Dedagroup
Publisher	Highlander Consortium
Contributors	Martina Forconi, Stefano Pezzi, Riccardo Valentini, Francesco Renzi, Damiano Giannelle, Gaia Laurin, Monia Santini
Type	Report
Language	EN
Rights	Copyright “Highlander Consortium”
Audience	Public
Requested deadline	M18

Highlander
High performance computing
to support smart land services

Index

1 Introduction

This project deliverable provides a high-level description of the data we are dealing with in the different Highlander DApOS from the point of view of the harmonization requirement.

Access, analyse and produce data homogenously saves time and money, avoids mistakes and helps the comprehension of the results. Standard specifications are designed to achieve these goals and Highlander project strongly leans on available international standards and, especially, on those regarding spatial information, furthermore, following the principles and directives of the European INSPIRE project.

In situ and remote sensing data are the two main types of data that are considered in the document: we will take into account the physical data format, the protocols and the service model used to access input data or publish results.

Using standards helps data harmonization, but it's not enough: almost every standard specification lives a certain degree of freedom especially at the semantic level, thus we'll show and explain some peculiar choices that fill these specification gaps.

Chapter 2 deals with IoT data coming covering three different scenarios (tree growth, animals health, wildfires), but managed in similar way. Chapter 3 covers the weather data and chapter 4 remote sensing images gathered by the multispectral sensor of Sentinel 2 and the SAR data of Sentinel 1 instrument.

2 IoT data harmonization

Internet of Things (IoT) consists of smart devices that communicate with each other. It enables these devices to collect and exchange data and offers excellent potential to collect time-series data for improving situational awareness.

The term “Internet of Things” (IoT) was first used in 1999 by British technology pioneer Kevin Ashton to describe a system in which objects in the physical world could be connected to the Internet by sensors. Ashton coined the term to illustrate the power of connecting Radio-Frequency Identification (RFID) tags used in corporate supply chains to the Internet in order to count and track goods without the need for human intervention. Today, the Internet of Things has become a popular term for describing scenarios in which Internet connectivity and computing capability extend to a variety of objects, devices, sensors, and everyday items.

Monitoring a single sensor alone might not offer much benefit; however, the monitoring of many sensors targeted at a linked causality brings valuable insights. If these sensors are connected to a network, one speaks of the Internet of Things (IoT), which is rapidly growing due to narrow production costs and huge potential in the field.

This offers the ability to measure, infer and understand environmental indicators, from delicate ecologies and natural resources to urban environments. The proliferation of these devices in a communicating–actuating network creates the Internet of Things (IoT), wherein sensors and actuators blend seamlessly with the environment around us, and the information is shared across platforms in order to develop a common operating picture.

IoT technology offers the possibility to transform agriculture, industry, and energy production and distribution by increasing the availability of information along the value chain of production using networked sensors. However, IoT raises many issues and challenges caused by a splintered sensor manufacturer landscape, data comes in various structures, incompatible protocols and unclear semantics. To tackle these challenges a well-defined interface, from where uniform data can be queried, is necessary. The **Open Geospatial Consortium (OGC)** has recognized this demand and developed the **SensorThings API standard**, an open, unified way to interconnect devices throughout the IoT.

The SensorThings API is a standard for the collection, storage and retrieval of time-series data.

The standard defines a model for sensor data and its metadata, as well as an interface for both data storage and retrieval. The service can be queried with powerful filters, including geospatial search possibilities. The goal of the SensorThings API is to offer a unified way for the usage of sensor-data and to facilitate the development of data-driven applications.

2.1 OGC:STA – SensorThings API

Sensor data have much to do with geographic data since they most of them comes from stations that have a specific location on the surface of Earth and/or refers to a specific area, point or, generally speaking, to a spatial feature. In situ observation data falls into this category as well as remote sensing data are very similar to satellite or aerial surveys images, even if the time dimension plays a very important role and so does the multidimensionality of the observed data.

There are also some kind of observations that are more difficult to assimilate to common geographic data, like data registered by moving sensors (like sounding balloon or ships or other vehicles on which sensors can be mounted) or remote sensing data that refers to volumes instead of surfaces.

For this reason, OGC has tackled the topic of sensor data in the larger context of geographic data, assembling a suite of standards called Sensor Web Enablement (SWE) and has tried to harmonize common concepts and to use the same protocols and encodings used for the better-known spatial services.

This suite has first hosted a conceptual model for observations and forecasts called O&M (Observations and Measurements) that become a ISO abstract specification and then a XML implementation that has been used to code the entities transferred by the Sensor Observation Service (SOS). Together with O&M, another coding specification has been released to describe sensors and, generally, computational processes used in a previous or in a following phase of the measurement.

The formal definition of SOS was defined likewise WMS, WFS or the other OWS (OGC Web Services), thus having a SOAP binding, an XML payload, a GetCapabilities operation for self-description and several other different operations.

SensorThings API (STA) is the evolution of the OGC:SOS Sensor Observation Service that addresses data access for the **Internet of Things** (IoT).

The vision of IoT is that of devices all over the world directly connected to the Internet to allow data retrieval and control.

This OGC standard is actually divided into two parts: the former dealing with **access** to data (Sensing Profile) and the latter is about **control** of devices (Tasking Profile) and in the context of Highlander we can ignore it.

SensorThings API provides a RESTful, JSON encoded API to retrieve data and metadata about 'things' that generate streams of data. The underlying information/resource model for the API uses Observation and Measures (O&M) and has been influenced by the other Sensor Web Enablement (SWE) standards.

The API follows patterns defined by the OData protocol. Table 1 shows the main differences between the previous standard SOS.

Feature	SensorThing	SOS
Encoding	JSON	XML
Architectural Style	Resource Oriented Architecture	Service Oriented Architecture
Binding	REST	SOAP
Pagination	$top/$skip/$next Link	Not supported
Pub/Sub Support	MQTT and SensorThings MQTT Extension	Not supported
OGC model link	Location entity	Confusion between feature and feature of interest
Insert new Sensors and Observations	HTTP POST	SOS specific interface: RegisterSensor() and InsertObservation()
Deleting Existing Sensors	HTTP DELETE	SOS specific interface: DeleteSensor()
Updating Properties of Existing Sensors or Observations	HTTP PATCH and JSON PATCH	Not supported
Deleting Existing Observations	HTTP DELETE	Not supported
Linked Data Support	JSON-LD	Not supported

Table 1 - MAIN DIFFERENCES BETWEEN STA AND SOS

2.1.1 Data model

SensorThings splits the O&M model across two classes: Datastream and Observation. Datastreams are the top-level class with a subset of the O&M Observation properties: observedProperty, resultTime, phenomenonTime. The observations property is analogous to result in the OM_Observation type, but in this case contains a set of Observation objects. Each of these is typed according to its observation type from O&M (Measurement, Geometry etc.).

The OGC SensorThings API data model not only covers plain sensor measurements, but also metadata like the unit of the measurement, a sensor description or a location. The metadata is connected to the original data stream.

The data elements are linked to each other enabling the user to find all necessary information of interest. The data model consists of eight entities, including properties and relations, which are shown in Figure 1.

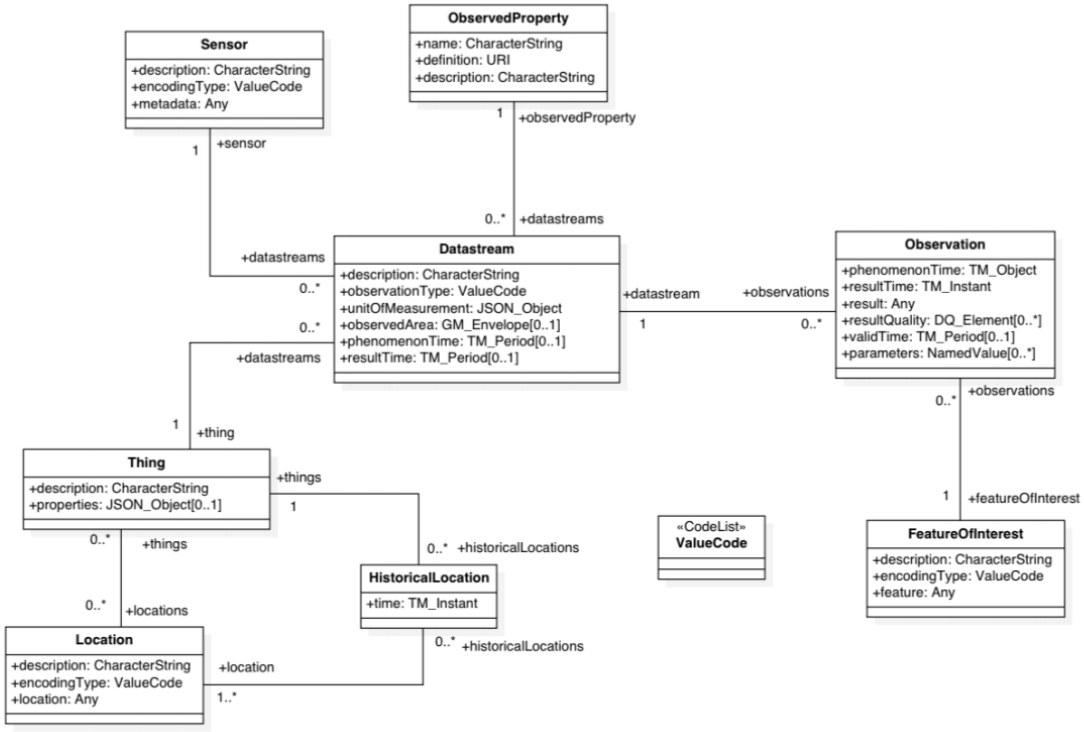

Figure 1 - SensorThings data model (Open Geospatial Consortium, 2016)

A *Thing* is a virtual or physical object. Depending on the use-case, the *Thing* can be the object being observed, like a river section, or the sensor platform, such as a satellite. *Location* describes the position of a *Thing*. A position can be described through geographic locations, encoded as points or areas. Symbolic locations, like a postal address, are possible, too. The *HistoricalLocation* is the link between a *Thing* and a *Location*, with the time indicating when the *Thing* was in a certain *Location*. A moving Sensor has several *HistoricalLocations*. A motionless *Thing* has none. This is defined through the SensorThings API, where the *HistoricalLocation* is created, when the *Thing* moves for the first time. A *Sensor* generates the data, which is described through the OGC SensorThings API Data Model. A *Sensor* can collect multiple *Datastreams*. A weather station for example can collect both temperature and humidity; in this

example, the *Sensor* weather station would have two entities *Datastream*, one for each temperature and humidity. Observation contains the measurement made by a Sensor. An *ObservedProperty* is a characteristic of the *FeatureOfInterest* that is observed by a *Sensor*. For example, the water level in a river, or its temperature. A *Datastream* unites *Observations* of an *ObservedProperty*, which were made by a *Sensor* and are linked to a *Thing*. The *FeatureOfInterest* can be the geographic area or location for which an *Observation* was made. This can be the same as the *Location* of the *Thing*, which is often the case for in-situ sensing. In the case of remote sensing, the *FeatureOfInterest* can be different from the location of the *Thing*, dependent on what is chosen as Thing. The *FeatureOfInterest* is a geographical point or a polygon encompassing an area or volume, usually encoded in the format GeoJSON. The relations between all these entities are described through the data model; this ensures finding all data entities that belong to another and only make sense in their own context.

2.1.2 REST

The OGC SensorThings API offers a RESTful interface for accessing the stored data. The REST programming paradigm is a well-known approach for realizing distributed systems. It is based on top of the Hypertext Transfer Protocol (HTTP), which forms the basis of the World Wide Web. REST is used for inter-machine communication and is widespread around web services. Alternatives are for example SOAP or Remote Procedure Calls.

The idea of REST was developed by Roy Thomas Fielding, published in his dissertation in (Fielding, 2000).

Fielding presents principles, which every REST-service must follow without suggesting how to implement them. The principles are described in the following. The first principle is the Client-Server Architecture, known from the World-Wide-Web. A server offers a service, which can be requested by a client. Through the usage of the widespread HTTP-protocol, a REST client implementation is available for nearly every programming language. The second characteristic of a RESTful service is its statelessness: every message sent to a REST service must contain all information needed to process this request. This brings two benefits: firstly, the service can be scaled according to the required usage. Secondly, it decreases complexity, since all information is summarized and no application state needs to be shared between two requests. Other than its alternatives, REST demands unified interfaces. This contains for example the addressing scheme. Every entity in REST has to be uniquely identifiable, which usually is implemented through Uniform Resource Locators (URLs). The representation of entities is often achieved through JSON. Requests to the server are transmitted via HTTP. Thus, the HTTP-methods GET, POST, PUT, PATCH and DELETE are used to interact with the server. Hereby, GET is used to

request information. To create a new instance of an entity the POST operation is used. By sending PUT (overriding a whole entity) or PATCH (overriding only provided attributes) messages, existing entities can be modified. DELETE is finally used to remove entities. For the SensorThings API, this means when entering the main URL of a SensorThings API server in a web browser, a GET request is issued to the server. The response of the server will contain a JSON file as shown in Figure 2.


```
value:
  0:
    name:    "Datastreams"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/Datastreams"
  1:
    name:    "MultiDatastreams"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/MultiDatastreams"
  2:
    name:    "FeaturesOfInterest"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/FeaturesOfInterest"
  3:
    name:    "HistoricalLocations"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/HistoricalLocations"
  4:
    name:    "Locations"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/Locations"
  5:
    name:    "Observations"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/Observations"
  6:
    name:    "ObservedProperties"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/ObservedProperties"
  7:
    name:    "Sensors"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/Sensors"
  8:
    name:    "Things"
    url:     "http://labora.dedagroup.it/FROST-Server/v1.1/Things"
```


Figure 2 - Response from the STA server base URL

The response contains all data model entities, as well as their URLs. Through these URLs, each individual on the server is addressable. Analogous to fetching data, an instance can be created, modified or deleted by sending a POST, PUT, PATCH or DELETE command to the appropriate URL. Since the database can contain a large amount of data, only a subset of all available data is returned for a GET command. This prevents both the server, as well as the client from an overload. Further data is sent upon request.

2.1.3 ODATA

The previous section showed how data can be retrieved and modified by using the REST interface. For many applications, it is not sufficient to retrieve all available data. Conditional upon the large variety of stored data, powerful and expressive filter mechanisms are required to receive the data of interest. In the SensorThings API, these are realized by applying the Open Data Protocol (OData), which is standardized by the Organization for the Advancement of Structured Information Standards (OASIS). It allows projections and filters similar to the Structured Query Language (SQL), which are specified as query strings in the URL. Projections can be done by naming the queried attributes in the *$select* parameter. For example, */Things?$select=@iot.id*, description will only select the id and the description of Things.

By passing the *$filter* parameter, it is possible to query for specific results. For example, */Observations?$filter=result gt 5* will return all observations that have a result value greater than 5. A wide range of filtering operators are supported; an exemplary list of supported functions is shown in Figure 3.

Comparison	Mathematical	Logical	String Functions	Geospatial	Date and Time
gt (greater than)	add (addition)	and	substringof(p0,p1)	geo.intersects(g1,g2)	now()
ge (greater equal)	sub (subtraction)	or	indexof(p0,p1)	geo.distance(g1,g2)	mindatetime()
eq (equal)	mul (multiplication)	not	trim(p0)	st_equals(g1,g2)	maxdatetime()
le (less equal)	div (division)		concat(p0,p1)	st_within(g1,g2)	date(t1)
lt (less than)	mod (modulo)			st_overlaps(g1,g2)	time(t1)
ne (not equal)					

Figure 3 - Exemplary list of functions that can be used with OData in the URL of requests

2.1.4 MQTT

Message Queuing Telemetry Transport (MQTT) is an open-source protocol standardized by OASIS that follows the publisher-subscriber pattern. Compared to other P&S protocols, MQTT is lightweight and minimises the network bandwidth and the device resource requirements, so it is suited for collecting the data of IoT-sensors.

By using this pattern, subscribers can register to a topic, which broadcasts information they are interested in. The information is initially provided by a publisher that sends a message to the corresponding topic as soon as it gets available. A message broker (included in a SensorThings API compliant server) takes care of the subscriptions and forwards the messages to all the subscribers registered to the topic. A topic is a string that can have several hierarchical levels, separated by a slash. Through this, a client receives only the information published within these topics. The naming of the topics is analogue to the URLs of the entities in the SensorThings API implementation.

An implementation of the SensorThings API offers, next to the RESTful HTTP interface, an additional MQTT interface. This interface is two-folded: that is, the server can act as a subscriber towards a sensor that publishes its observations, collect these data and store them. On the other side, a client, that could be a processing component, can subscribe to a server topic corresponding to this *datastream* to get the observations and, for example, trigger some action when a threshold is reached. It's not mandatory to use MQTT on both side of the data flow, i.e. the observations could be ingested into the server via HTTP channel.

A possible use case, suitable with the Natural Parks scenario, could be that of the battery voltage measures that each TT+ device sends to the server, along with the other observations, and a process that listens to the corresponding MQTT topics and that triggers an alert, like an email, as soon as the voltage falls under a prefixed value. Since TT+ data do not arrive into the server with a very high frequency, here the plus is mainly that clients do not have to poll server to gather new data: it's the MQTT protocol that implements a push model.

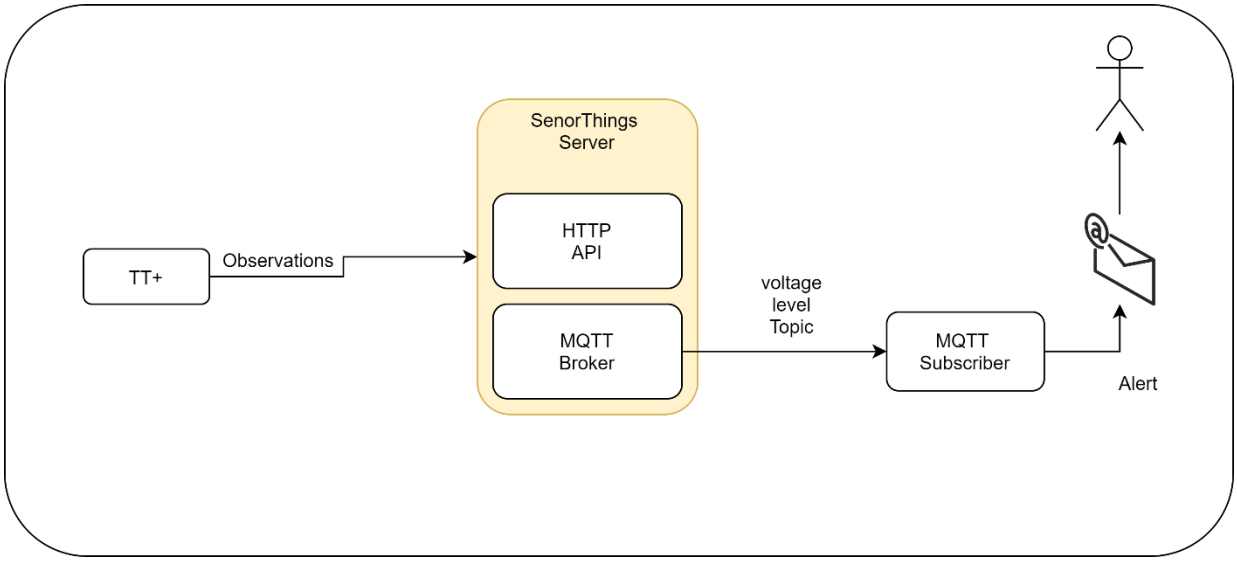

Figure 4 - Simple use of the MQTT broker

2.2 Natural parks IoT data

The goal of this DApOS is to analyse tree functional biodiversity by collecting a bunch of biological indicators of the vital functions of trees in a forest environment. These data will be collected by two clusters of **Tree-Talker+** sensors. Each cluster is formed by 25 TT+ devices and monitors respectively a group of beeches and one of spruces; both are deployed in Val Canali.

Figure 5 - TreeTalker+ device on the left image (battery pack is the lower box). TT cloud transmitter with an additional solar panel in the top right image. A portion of a deployed TT+ cluster in the bottom right image.

Tree-Talker device version "TT+" measures simultaneously important individual tree scale eco-physiological parameters as well as some additional ecosystem-related variables.

Key parameters are:

- Tree radial growth, as an indicator of photosynthetic carbon allocation in biomass
- Sap flow, as an indicator of tree transpiration and functionality of xylem transport
- Xylem moisture content as indicator of hydraulic functionality
- Light penetration in the canopy in terms of fractional absorbed radiation
- Light spectral components related to foliage dieback and physiology
- Tree stability parameters to allow real time forecasts of potential tree fallings.
- Additional parameters such as air relative humidity and air temperature will be also monitored at high frequency to have comparable time scale between abiotic parameters and short-term plant responses.

The Tree-Talker devices are connected by using LoRa protocol of radio communication to the gateway **TT-Cloud**. TT-Cloud can receive and store data up to 48 devices in one cluster (a maximum of 20-30 is suggested in practice to avoid data collision) and data transmission is

typically set at hourly frequency although customizable. The TT-Cloud is in turn connected to the internet via GPRS network and sends data to a computer server. TT-Cloud can cover devices spread approximately in a radius of 3 kilometers provided the area is reasonably flat without many obstacles. The direction of TT-Cloud should be toward watching other Tree-Talker; however, an external antenna will facilitate the signal coverage.

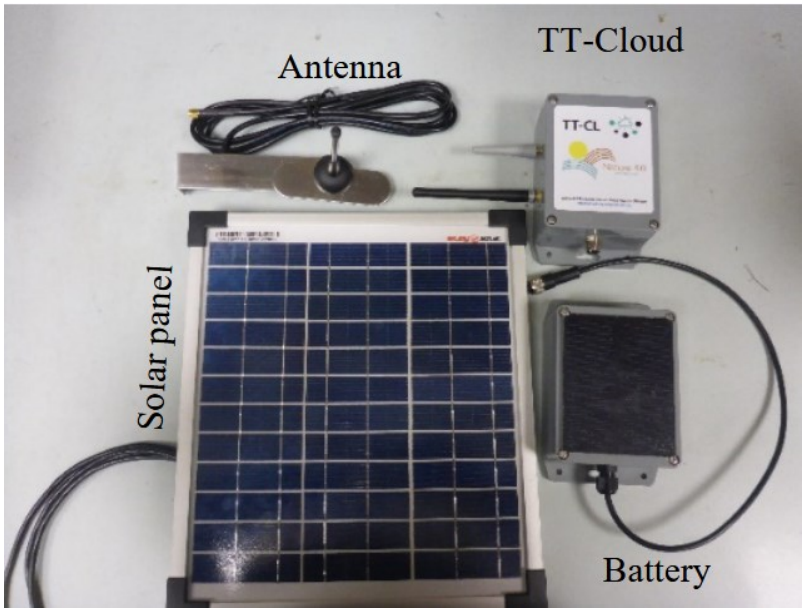

Figure 6 - The TT-cloud gateway

2.2.1 Mapping TreeTalker network to SensorThing entities

The Tree-Talker scenario maps its objects into the STA sensing entities quite easily, but there are some peculiarities that must be highlighted and that will be illustrated in the next paragraphs.

Note that we will try to remain inside the standard specifications of STA because this is the key of the harmonization and it ensures the maximum level of interoperability.

We will also take advantage of the new features of the standard introduced with **version 1.1**: all entities (except for HistoricalLocation) now have a field of the type JSON Object that can be used to store additional information about the object. In version 1.0 such a field already existed for the Thing entity (named *properties*), and for the Observation entity (named *parameters*).

These fields can be used in filters and are completely customizable since the JSON dictionary implements a *key-value pair* paradigm.

		Numerosity			
STA Entity	**TreeTalker Entity**	**# TT+**	**# TT-R**	**# TT-CL**	**# Total**
Thing	TreeTalker device	40	1	2	43
Location	TreeTalker's location	40	1	2	43
Sensor	Growth sensor, gravity sensor, spectrometer, etc	7*40 = 280	3*1 = 3	4*2 = 8	291
ObservedProperties	Battery ADC, Stem Radial Growth, Air Temperature, etc	22			22
Multi/DataStream	Time series identified by: OP, T and Sensor.	17*40= 680	6*1 = 6	6*2 = 12	698
FeatureOfInterest	Tree	40	1	2	43
Observation	Time and value of observation	1 per hour per Multi/Datastream			698 * hour of observation

Table 2 - Mapping of entities and numerosity of instances

2.2.2 Things

The **TreeTalker+** (TT+) device fits into the concept of **Thing**, that STA data model describes as a sensor platform or a station that is capable of being identified and integrated into communication networks. A TT+ is able to transmit observations to a server and contains several **Sensors,** that is, devices that respond to a physical stimulus producing a measure of its intensity or other quantitative parameter.

So, in our mapping there will be a Thing for each of the TT+ boxes installed and one for the unique TT-R (a similar device that hosts only environmental sensors). Finally, since the TT-cloud concentrator is also capable of doing measures that are sent along with the others observations (the strength of the transmission links with the connected TT+s and of the downlink with the GSM Base Transceiver Station and some other operational parameters), even this device is modelled as a Thing.

2.2.3 Sensors

In our mapping, a **Sensor** is the single device contained in one of the TT* boxes that observes a certain biological or physical parameter. We may have simple and complex sensors (meaning with "simple" a sensor that observe a single parameter) and, according STA model, also

processes that executes software algorithms on raw measures to calculate derived values can be considered as sensors or part of them.

In our case, inside a TT+ deployed in-situ, we have mostly physical electronic devices with or without embedded computing components that produce the values of only the parameters they measure directly (like a frequency instead of a humidity value) that is the values of an electrical parameter (digital or analogic) without finally transforming it into the measure of the correlated physical/biological phenomenon.

The procedures needed to obtain the physical/biological parameter of interest must then be executed on these raw observations by specific processes that run on the server. Some of them will be embedded in the **ingestion processes**, since the functions they implement take as input only the instantaneous raw values coming from the sensors; others apply an aggregation of values over time and thus it would be much more complicated calculate this function along the ingestion data flow and so will be implemented as **batch scheduled processes** once the raw data have been stored in the DB. This is the case of "SAP flow" parameter, but we'll analyze it in detail later.

For the sake of simplicity, even though the processing of the raw signals is temporally and spatially separated from the in-situ devices, we will consider these processing components fully part of the related sensors. The O&M specifications (Observations & Measurements) to which the STA standard refers for the data model, have foreseen the required time attributes to differentiate between the moment an observation is made and when it has become available, and we'll use these attributes to properly characterize raw and calculated observation.

Name	Description	Metadata	TT+	TT-R	TT-CL
SapFlowSensor	Modular probes for sap flow	Reference and heated temperature probes (±0.1 °C). Thermistors manufacturer: Murata Electronics. Model: NCU18XH103F6SRB.	X		
StemHumiditySensor	Stem humidity	Capacitive sensor MicroPCB (20x3x2) mm with copper plates.	X		
GrowthSensor	IR radial growth sensor	Infra-red distance sensor (±100 µm) Manufacturer: SHARP. Model: GP2Y0A51SK0F.	X		
Battery	Long life batteries or solar panel	Battery: 4 Li-Ion batteries + solar panel (3.7 V)	X	X	X
GravitySensor	Accelerometer for tilt angle determination	Accelerometer (± 0.01°) Manufacturer: NXP/Freescale. Model: Si7006	X		

Spectrometer	Twelve bands spectrometer	Radiometer-12 spectral bands (450, 500, 550, 570, 600, 610, 650, 680, 730, 760, 810, 860 nm) (± 10 nm or ±20 nm). Manufacturer: AMS. Model: AS7262 (Visible range), AS7263 (Near Infra-Red range)	X	X	
WeatherSensor	Air temperature and humidity sensor	Thermohygrometer (±0.1 °C ; ±2 %). Manufacturer: Silicon Labs. Model: MMA8451Q	X	X	
ModemRouterLoRa	Modem/router LoRa protocol	868 MHz with external antenna			X
ModemGPRS	Modem GPRS				X
FlashMemory	Flash memory for data storage	16 Mbyte			X

Table 3 - Identified Sensors with correspondent type of Thing in which you can find them

2.2.4 ObservedProperties

An ObservedProperty is an aspect of physical phenomenon that a Sensor can observe and measure. At first, one could imagine that in our scenario the only ObservedProperties we deal with are those of strict interest for the analysis domain, that is the tree vital, biological and structural parameter (sap flux, xylem moisture, trunk radial growth …), but we must take into account even the sensor platform's indicators like battery voltage or the radio transmission power to implement some quality checks on data and to monitor the operations and eventually schedule some maintenance interventions. Furthermore, at this stage of the project[1], is important to store the raw and instrumental observations because, if some issues should emerge from the data analysis regarding the adopted algorithms, we could always reapply to the raw data the adjusted transformations.

Thus, we could identify three types of the **ObservedProperties:**

- **raw** properties, numbers from digital electronics or analogue electrical values that do not measure directly a domain physical/biological phenomenon, neither an operational physical phenomenon[2].

[1] When the analysis of the results will confirm the various algorithms used to produce calculated observations, we could get rid off the raw or instrumental observations and do not collect them any more for further campaign of observation.
[2] The result of the IR sensor that send an impulse toward the surface of the trunk.

- **instrumental** properties, a physical, but meaningless parameters[3] in terms of domain of analysis; ,
- **biological** properties, the aspect of the real biological phenomenon[4] we are interested in. To keep it simple, we will consider in this category even meteorological parameter like air temperature and humidity.

We will use the *properties* attribute to tag ObservedProperties with their category.

```
{
  "@iot.id": "FreqUnheatedSensor",
  "name": "FreqUnheatedSensor",
  "description": "Frequency measurement on the unheated sensor",
  "definition": "https://www.unitus.it/iot/terms/observedproperties/ FreqUnheatedSensor ",
  "parameters": {
                    "category": "raw"
                }
}
```

Snippet 1 -Sample instance of a raw ObservedProperty

```
{
  "@iot.id": "TrunkSurfaceDistance",
  "name": "TrunkSurfaceDistance",
  "description": "Distance of the trunk surface from a point kept at a fixed distance from
the xylem",
  "definition": "https://www.unitus.it/iot/terms/observedproperties/TrunkSurfaceDistance",
  "parameters": {
                    "category": "instrumental"
                }
}
```

Snippet 2 -Sample instance of an instrumental ObservedProperty

```
{
  "@iot.id": "StemRadialGrowth",
  "name": "StemRadialGrowth",
  "description": "Growth of the trunk measured on a radius",
  "definition": "https://www.unitus.it/iot/terms/observedproperties/StemRadialGrowth",
  "parameters": {
                    "category": "biological"
```

[3] For instance, the "sap flow" is derived from the temperatures of a heated sensor and of an unheated one, measured at the start and at the end of a specific interval of time: these temperatures are yet physical parameters, measured in degree Celsius, but meaningless if considered by themselves.
[4] The Radial Growth of Trunk or the Sap Flow are some of these properties.

```
                }
}
```

Snippet 3 -Sample instance of a biological ObservedProperty

Biological ObservedProperties often must be calculated by some function applied by a computing component that can be fully integrated with the device or not.

In this latter case, that of a distinct elaboration phase, we must have available the raw or the instrumental properties. When the calculated properties are the result of a simple and invertible function you could avoid storing the input values because, if you need them, you can easily obtain them back by reversing the calculation. This is the case of the battery voltage or the air temperature. Notwithstanding this consideration, we will store all types of raw observations except those to which only a change of unit of measure is applied.

Important: some parameter will not be mapped as ObservedProperty, since they are not *per se* interesting, but they will be attached to an observation for which they describe an environmental condition or bring additional information. In these cases, when creating the instance of the related Observation, we will attach the value of these parameter as a "parameters" attribute, that is key/value pair of a JSON object assigned as value of this attribute of the Observation.

The standard deviation of each acceleration falls into this kind of parameter and so does the battery voltage for Stem Radial Growth or the probe temperature for the frequency measurement.

```
{
  "phenomenonTime": "2020-05-12T13:00:00.000Z",
  "resultTime": "2020-05-13T12:00:00.000Z",
  "result": 32.0,
  "parameters": {
                    "voltage": 9.87
                }
}
```

Snippet 4 - Sample observation with parameters attribute

In Table 4 there is the list of ObservedProperties that we have identified in the system; for these properties there will be incoming streams of Observations from the various Things that will be stored in the STA repository.

In this table the unit of Measure (UoM) is associated to the property: STA model attaches UoM to DataStream and the same ObservedProperty could be associated to different DataStreams

with different UoM (i.e. you could store the very same observation of Air Temperature as two Observations, one measured in °F and one in °C). Since this is not our intention, for the sake of understanding we show here the information of UoM, but we will strictly follow the STA data model.

Name	Description	Category	TT+	TT-R	TT-cloud	Calc'd	UoM	Dependencies
IRSensorDistance	Reading from the infra-red pulsed distance sensor positioned at few centimetres away from the tree trunk's surface	Raw	X				d.n.	
TrunkSurfaceDistance	Trunk surface distance derived from raw reading of infra-red pulsed distance sensor	Instrumental	X			X	cm	IRSensorDistance BatteryADC (parameter)
StemRadialGrowth	Growth derived from trunk surface distance	Biological	X			X	cm	TrunkSurfaceDistance
TempUnheatedSensor	Temperature on sensor not heated (reference probe) at the beginning of measurement	instrumental	X				°C 10^{-1}	
TempHeatedSensor	Temperature on heated sensor at the beginning of the measurement	instrumental	X				°C 10^{-1}	
TempUnheatedSensorEnd	Temperature on sensor not heated at the end of measurement	instrumental	X				°C 10^{-1}	
TempHeatedSensorEnd	Temperature on sensor heated at the end of measurement	instrumental	X				°C 10^{-1}	
SapFlow	Sap flow according to the thermal dissipation method (TDP) of Granier	Biological	X			X	g m^{-2} s^{-1}	TempUnheatedSensorEnd TempHeatedSensorEnd
FreqUnheatedSensor	Frequency measurement on the unheated sensor	Raw	X				Hz	
XylemMoistureContent	Estimated amount of stem humidity	Biological	X			X	%	TempUnheatedSensor (parameter) FreqUnheatedSensor
AirRelativeHumidity	Air relative humidity	Biological	X	X			%	
AirTemperature	Air temperature	Biological	X	X		X	°C 10^{-1}	Change of UoM (°C 10^{-1} → °C)
TreeAcceleration	Acceleration and variance along the (Z,Y,X) dimensions	Raw	X				d.n.	Variance (parameter)

Co-financed by the Connecting Europe Facility of the European Union

TiltAngle	The angle between the gravity vector and the z-axis	Biological	X			X	degree	TreeAcceleration
TransmissionSpectrum	Canopy light transmission in 12 bands: 610, 680, 730, 760, 810, 860, 450, 500, 550, 570, 600, 650 (unit unknown)	Instrumental	X			X	[unknown]	Linear mapping from raw reading
IncidentSpectrum	Canopy incident light in 12 bands: 610, 680, 730, 760, 810, 860, 450, 500, 550, 570, 600, 650 (unit unknown)	Biological		X		X	[unknown]	Linear mapping from raw reading
BatteryADC	Battery analog to digital converter	Raw	X	X			d.n.	
BatteryLevel	Battery level	Instrumental	X	X	X		mV	BatteryADC (only for TT+, TT-R)
AccumulatedRecords	Accumulated records in memory	Instrumental			X		pure number	
NumberOfRecordsToSend	Number of records that still needs to be send to the server	Instrumental			X		pure number	
GSMFieldLevel	GSM field level	Instrumental			X		[unknown]	
RSSIRadioSignalStrenght	RSSI received radio signal indication strength between TT+ & TT-CL (21)	Instrumental			X		dBm	

Table 4 – ObservedProperties. Rows are grouped by sensor.

2.2.5 DataStreams

A DataStream represents a time series of observations registered by a specific Sensor, made available by a specific Thing and regarding a specific ObservedProperty.

2.2.6 Locations

A Location in STA data model is the geographical position of a Thing. This for most of the in-situ sensor scenario, and thus also for our case. A point geometry type is the most appropriate type of location.

2.2.7 FeatureOfInterest

This entity represents the real-world object that is the target of the Observations. Not always is possible to identify such an object: if you're remote sensing an area from an aircraft or a satellite, the feature of interest is rather a geographical area than a physical object, or if you have an anemometer even this spatial meaning could have fading borders.

In our case, the individual tree to which the sensor is attached is indeed the FeatureOfInterest. and to this object we can attach information like the tree species, its height, the estimated age…

TT-R and TT-cloud stations are as well attached to trees, but only serving as a physical support because the sensors are not measuring their biological/physical parameter, but weather or transmission related parameters. So, the FeatureOfInterest for the TT-cloud could be the polygonal area that contains the TT+s linked to it, calculated as the convex hull of their locations, as shown in Figure 7.

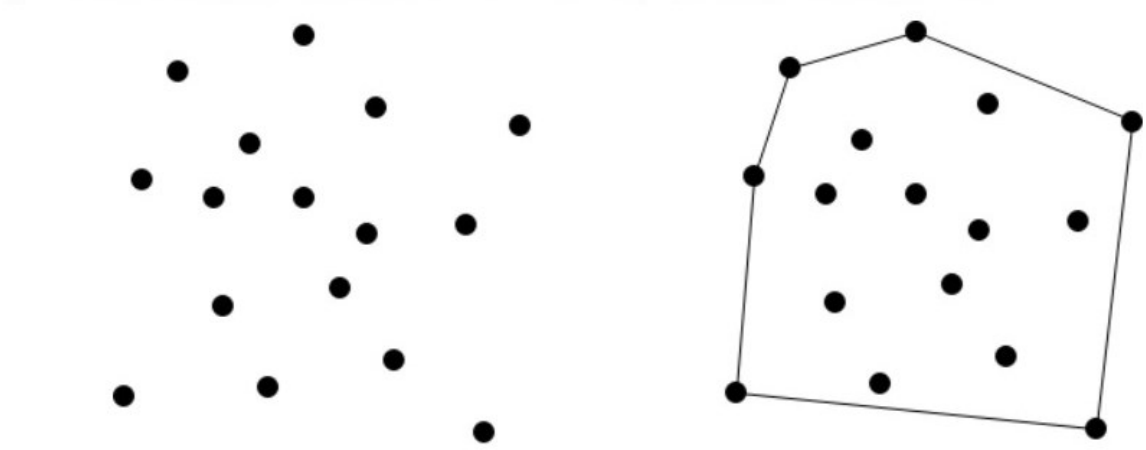

Figure 7 - Convex hull of locations of a TT+ cluster

For a TT-R station, since it measures weather conditions that can be considered similar up to a certain distance, we could think of a minimum circle that includes all the deployed stations with centre in the TT-R location.

2.2.8 Identifier

Defining nomenclature rules for the data model instances is quite important because the process of initial creation of the database and the following ingestion of observations depend on these rules.

Our choice has taken into account even the way data are made available by the TT* devices and the constraints of the STA data and service models.

To recap, our boundary conditions were:

- each data row coming from the fields stations contains the serial number (SN) of the TT+ that produced it.
- there's not a preliminary identification by means of a registry number of the trees subject to the monitoring; only their position is acquired via GPS and registered. These coordinates are not recorded on the device but are kept externally.
- When inserting observations, STA data and service models force you to indicate the Datastream they belong to together with the related FeatureOfInterest.

The identifiers could be:

- managed by server
- managed by client
- hybrid mode: both server and client can define identifier.

The last configuration is the one that best fit our needs: in fact, we can use the client-defined identifiers for those entities that are stable or that change very rarely and that are inserted into the database during its start-up phase, that is:

- Location
- Sensor
- Thing
- Datastream
- ObservedProperty
- FeatureOfInterest

Any instance of these entities needs to be identified properly by a user because almost every use case are query like "which ObservedProperty is collected by Sensor X?" or "which Sensors are contained in Thing Y?" or "give me the Observation gathered by Sensor X regarding

ObservedProperty Z" thus their identifiers should better be "meaningful identifier" that a human can guess knowing some characteristics of the object he/she's looking for.

Observations instead are automatically produced by sensors and rarely need to be identified individually by the user, so we'll let the system assign an identifier.

Said this, the **SN of the TT+** device becomes fundamental for the construction of the identifiers of the following entities:

Entity	Pattern	example
Location	Loc_<SN>	`Loc_21990671`
Sensor	<TypeOfSensor>_<SN>	`ModularProbes_21990659`
Thing	TT_<SN>	`TT_21990699`
DataStream	DS_<ObservedProperty>_<SN>	`DS_TiltAngle_21990668`

Table 5 - Rule for naming static entities that depend on Things

The remaining "static" entities are independent from the TT+ devices, that is an ObservedProperty observed by a Sensor belonging to a TT+ Thing, but it can be observed also by a totally different Sensor belonging to another Things located at the opposite side of the world. The FeatureOfInterest is independent as well the tree has been there before the Thing was installed and will continue to exist after the Thing will be removed. For this kind of entities, the naming rule are quite different:

Entity	Pattern/rule	example
ObservedProperty	meaningful name of the property, CamelCase formatted	`TreeAcceleration`
FeaturesOfInterest	<area>_<tree species>_<sequence>	`paneveggio_fir_0001`

Table 6 - Rule for naming static entities indepedent from Things

Finally, for the Observations, we'll let the server create the. That's the type of identifier that will be used for Observations.

Entity	Pattern/rule	example
Observation	UUID	`77957484-9162-11eb-8243-9f2433bacadb`

Table 7 - Rule for dynamic entities Observations

IMPORTANT: during the ingestion phase, the process that implements the automation will detect the SN inside the Observations data coming from a cluster of TT+ devices. With this SN, the process will identify the correct DataStreams to which append the various Observations, but it needs to know the correct FeatureOfInterest too. There's no manner to infer this information from the Observations flow and, therefore, one should store the Thing-FoI pairing is some external data structure that will be looked-up by the process. If a TT+ is substituted completely (it's something that may happen), a new Thing, new Sensors and new Location shall be created (and a new DataStream, too), but the ObservedProperties and FoI shall remain the same: thus the pairing information shall be updated as well.

2.2.9 Versioning of sensor devices

It's possible that during its period of duty, a TT+ device is subject to a maintenance intervention aiming to an upgrade of a contained Sensor in terms of hardware or firmware. Both type of upgrade can be backward-compatible, but also could break this compatibility, meaning that observations streaming from the sensor after the intervention could not be comparable anymore with those collected before.

In both cases we must trace this upgrade event and be able to obtain information of the sensor version that collected an observation. On the other side, when the upgrade maintains backward compatibility, it should be easy to treat the relative observation time series as one.

Another requirement is that the usual procedure adopted till now is to version the whole TT+ box, i.e.: a firmware update that improve or resolve some issue only for the accelerometer, results in a new version of the TT+. In order to keep track of these changes over time, a **Datastream** called "version" will be created and its **ObservedProperty** will be the TT+ version. The observation will also contain the upgrade date. Searching among these observations the user can get the validity period of a particular version or the version that was active at a certain time.

To recap:

- If a **backward-compatible** upgrade is made to one or more Sensors or part of them, a new observation will be added to version Datastream with the new version installed, the date of installation and the information about the new version.
- If **non-backward-compatible** upgrade is made, together with the above updates, a new Sensor must be created along with a new DataStream added to the Thing and the old Sensor must be ceased.

2.3 Forest fire predictions and controls

The aim of this DApOS is to produce maps of fire risk and of variables relevant to fire prevention and control, in selected forested natural parks of the Puglia regions, based on climate, remote sensing, and proximal sensing data properly integrated thanks to machine learning and Big Data analysis tools. Specifically, a regional network of TreeTalker and TreeTalkerFire in-situ stations allow the characterization and continuous monitoring at tree level of selected parameters that are acknowledged to be important in triggering and fostering forest fires. In total 240 sensors are distributed in 4 regional parks area (120 are TreeTalker devices and 120 TreeTalkerFire devices). The TreeTalkerFire variables are:

- Air temperature,
- Relative humidity,
- Foliage Temperature,
- CO2 concentration,
- O3 concentration,
- PM2.5 and PM10 concentration,
- Presence of Flame.

All the previous considerations made for TreeTalkers regarding management of data are also valid for TreeTalkerFires. The main differences between the two systems are the sensors mounted on the devices and the way the data will be used. Fire prevention needs an almost real time monitoring and the data should be treated accordingly, this consideration explains the presence of a variable called “Presence of Flame” that is a flag triggered in case of fire. However, the use of the data does not regard their storage but their analysis.

2.4 IoT for animal wellbeing

Animaltalker device is currently under development but the managing of the data can be described knowing its characteristics.

An Animaltalker is a device that can receive data via Bluetooth from different sensors called animal buttons. Each button is composed of a sensor, a battery and a Bluetooth microchip with its antenna. It sends the data collected in a string format using a BLE (Bluetooth Low Energy) service. The Animaltaker gathers all the strings with their IDs and makes them available over the internet. The buttons under development now are:

- A skin temperature button;
- A movement button (using a 3 axis accelerometer);
- An under skin temperature button;
- A heart rate button.

The AnimalTalker has a GPS module, an environmental temperature and humidity sensor, a NB-IoT/4G/5G module to allow internet connection and the possibility to mount a LoRa module to work also in places where obtain a good internet connection is particularly challenging.

2.4.1 Mapping AnimalTalker network to SensorThing entities

Most of the comments made for TreeTalkers are valid for the AnimalTalker network. There is also the will to use the same data structure for all the networks of the project. A major difference between the two systems is that the animal position will change over time so the "HistoricalLocation" entity will be used to keep track of the subsequent animal GPS coordinates.

STA Entity	AnimalTalker Entity
Thing	AnimalTalker device
Location	Animal last GPS coordinates
HistoricalLocation	Previous animal GPS coordinates
Sensor	Buttons and AnimalTalker sensors

ObservedProperties	Movements, skin temperature, under skin temperature, …
Multi/DataStream	Used if a particular button will send related multiple data
FeatureOfInterest	Animal
Observation	Time and value of observation

Table 8 – AnimalTalker mapping of entities

2.4.1.1 Things

The AnimalTalker will be considered a **Thing** while the buttons will be Bluetooth **Sensors**. If the device will work using LoRa protocol, the LoRa gateway will also be considered a **Thing**.

2.4.1.2 Sensors

As stated above, in our mapping a **Sensor** will be the single Bluetooth button but also the AnimalTalker itself will host some sensors such as the environmental humidity and temperature sensor and even the LoRa gateway will send some useful data. The buttons will have a microchip more powerful than the TreeTalker one. Therefore, some sensors will present on their software suitable algorithms to perform on chip the analysis of measurements. Further analysis could be performed server side such as the ones that require an aggregation of values over time and for this reason must be analysed using batch scheduled processes. All the data derived from the same physical electronic device will be considered part of the related sensor.

Name	Description	Button	AnimalTalker	LoRa gateway
SkinTemperatureSensor	Temperature sensor placed on animal skin	X		
UnderSkinTemperatureSensor	Temperature measured by an under skin RFID sensor	X		
MovementSensor	High frequency accelerometer to detect specific animal movements	X		

HeartRateSensor	Optical heart rate sensor	X		
WeatherSensor	Air temperature and humidity sensor		X	
Battery	Battery level	X	X	X
ModemRouterLoRa	LoRa protocol Modem/router			X
ModemGPRS	NB-IoT/4G/5G module		X	X

Table 9 - Identified AnimalTalker sensors with correspondent type of Thing in which you can find them

2.4.1.3 ObservedProperties

The measured physical phenomena are related to sensors described in the previous paragraph. We will always save, if possible, the raw data because they represent the true measured value, thus the reference one. However, taking into account some sensors such as the 20 Hz accelerometer, the raw data should be sent only during the first stage of the project to allow the analysis of the phenomenon measured. In every day conditions, the amount of data collected would be too much to be made available over the internet (the value of this kind of data changes quickly). For this reason, data should be saved locally and an algorithm should be uploaded on the button to send only the final output. The algorithm could also monitor the changes in the measure of interest and start the data collection when a particular condition is reached. Some performed measurements can be more interesting when related with other parameters. In these cases, the main information will present the value of the secondary one as a "parameter" attribute, which is a metadata of an **ObservedProperty**.

In conclusion, also for AnimalTalker we will use three types of the **ObservedProperties:**

- **raw** properties, numbers from digital electronics or analog electrical values that do not measure directly a domain physical/biological phenomenon, neither an operational physical phenomenon.

- **instrumental** properties, a physical, but meaningless parameters in terms of domain of analysis; ,
- **biological** properties, the aspect of the real biological phenomenon we are interested in. To keep it simple, we will consider in this category even meteorological parameter like air temperature and humidity.

In Table 10 there is the list of ObservedProperties that we have identified in the system; for these properties there will be incoming streams of Observations from the various Things that will be stored in the STA repository.

Now, we do not fully know all the analysis that will be performed over data collected and if they will be implemented server side or button side. Thus, we listed on Table 10 some examples of the data that will be saved on the database but they should not be considered conclusive.

Name	Description	Category	Button	AnimalTalker	LoRa gateway	UoM
SkinTemperatureAnalog	Output of insulated surface temperature sensor placed on a meaningful position.	Raw	X			d.n.
SkinTemperature	Temperature of animal skin derived from the surface temperature sensor.	Instrumental	X			°C
BodyTemperature	Animal Core temperature obtained using skin temperature and air temperature and humidity.	Biological	X			°C
UnderSkinTemperature	Temperature measured by an under skin RFID sensor.	Biological	X			°C
MovementAnalysis	Movement readings performed by a high frequency accelerometer to detect particular movement patterns of the animal.	Instrumental	X			
HeartRateOptical	Measures obtained by an optical heart rate sensor that can be related to the animal heart rate.	Raw	X			d.n.
HeartRate	Animal heart rate	Biological	X			bpm
AirRelativeHumidity	Air relative humidity	Biological		X		%
AirTemperature	Air temperature	Biological		X		°C
BatteryLevel	Battery level	Raw	X	X	X	
SupplyVoltage	Battery supply voltage	Instrumental	X	X	X	mV
GSMFieldLevel	GSM field level	Instrumental		X	X	
RSSIRadioSignalStrenght	RSSI received radio signal indication strength between TT+ & TT-CL (21)	Instrumental			X	dBm

Table 10 – AnimalTalker ObservedProperties. Rows are grouped by sensor.

Co-financed by the Connecting Europe Facility of the European Union

2.4.1.4 DataStreams

A DataStream represents a time series of observations registered by a specific Sensor, made available by a specific Thing and regarding a specific ObservedProperty.

2.4.1.5 Locations and HistoricalLocations

Positioning is particularly important in this project because animals move during the day. For this reason, the device has a GPS module that can provide geographic coordinates that will be saved in **Locations**. Every time the field will be updated, the old position will be stored in **HistoricalLocations**.

2.4.1.6 FeatureOfInterest

This entity represents the real-world object that is the target of the Observations. The AnimalTalker **FeatureOfInterest** will be the animal itself. This field is particularly useful to identify the animal and to storage its characteristic like breed, feeding, age, etc.

2.4.2 Identifier

Identifiers are IDs used by the database to identify univocally an entity. The IDs can be chosen by the database in an automatic way or by the user. We choose to use the latter because the IDs are used to create, update and delete an entity so select them following a method that is easy to understand and that can be automated is of primary importance. AnimalTalker identifier and the name of the entity will compose the database entities identifiers. The reason behind this choice is that each string sent by every device to the server will start with the device id and the users will know the ids of their device so using the protocol previously described will make the system managing easy. Only the **ObservedProperties** will be identified using an id chosen by database itself because the measurements values should not be modified.

2.4.3 Versioning of sensor devices

A change in AnimalTalker version will be treated in the same way of a change in TreeTalker version.

3 Harmonization of climate reanalysis and projections

The climate research community recognizes since years file formats, standards and conventions favoring further harmonization and enabling interoperability, among climate-related products but also with other types of data. Climate data are assumed already harmonized as any variable is provided under the same format. In this paragraph, the formats, standards, and conventions used in HIGHLANDER for climate model generated data are described.

3.1 File format

One of the first aspect to consider in climate modelling outputs is the data format. Climate research consider 3 generic categories: *GRIB*, *NetCDF* and *HDF*. All of these formats are portable (machine independent) and self-describing, i.e. they can be examined and read by the appropriate software without the user is expected to know the file's structural details. Further, additional information about the data, called "*metadata*", may be included in the file, e.g. textual information about each variable's contents and units, or numerical information describing the grid type, the coordinates (*e.g.*, time, level, latitude, longitude). The most recent and used data formats are as follows:

netCDF4: Network Common Data Format (Version 4.x)
GRIB2: GRIdded Binary (Edition 2)
HDF5: Hierarchical Data Format (Version 5.x)

NetCDF (*Network Common Data Form*) is designed to facilitate access to array-oriented scientific data and is the format most commonly used for climate model generated data. NetCDF contains a header which describes the layout of the rest of the file, in particular the data arrays, as well as arbitrary file metadata in the form of name/value attributes. As said, the additional information about a file or variable is commonly called "metadata".

There are two versions of netCDF-{3,4}. The netCDF-3 data model was used for many years and is often referred to as netCDF-classic. However, as datasets became larger, the grids more complicated, and user desires for more flexibility developed (e.g. support compression, string variables or parallel processing) the netCDF-4 (nc4) was created.
There are three different *software* categories used for climate data processing and visualization: (1) compiled languages (*eg.*, fortran, C, C++); (2) command line operators and viewers (NCO, CDO, ncview, panoply); (3) interpreted languages (NCL, GrADS, Ferret, R, Generic Mapping Tools (GMT), Perl Data Language (PDL) , Python [CDAT/PyNIO/PyNGL/Numpy/matplotlib], and the commercial products Matlab, IDL and, to a lesser extent, PV-Wave).

The typical NetCDF file has several components: (a) *dimension names*; (b) *dimension sizes* of the dimension names; (c) the *variables* on the file which often include additional information about each variable and temporal/spatial coordinates; and (d) *global attributes* which contain information about the file's contents, authors, post-processing history etc.

To enable spatio-temporal referencing, the NetCDF file contains coordinate variables which are defined as one-dimensional variables with the same name as a dimension. Coordinate variables should not have any missing data (for example, no **`_FillValue`** or **`missing_value`** attributes) and must be strictly monotonic (values increasing or decreasing).

The following is an example of a typical NetCDF file created in HIGHLANDER from Task 4.1 (Downscaling of ERA5 reanalysis, VHR-REA_IT, see Deliverable 4.1), referring to hourly 2-meter air temperature for 1989 (i.e. 8760 time steps, 24 hours by 365 days). In the example, **time**(*time*), ***rlat***(*rlat*), and **rlon**(*rlon*) are classified as coordinate variables while **T_2M**(*time*,*rlat*,*rlon*) is classified as variable. In this case coordinates are in rotated pole grid, represented by rlat and rlon (see explanation in the next paragraph).

```
netcdf file:/C:../T_2M_HLDRea_002_1hr_1989.nc
{
  dimensions:
    time = UNLIMITED;   // (8760 currently)
    bnds = 2;
    rlon = 535;
    rlat = 680;
  variables:
    double time(time=8760);
      :standard_name = "time";
      :long_name = "time";
      :bounds = "time_bnds";
      :units = "seconds since 1988-01-01 00:00:00";
      :calendar = "proleptic_gregorian";
      :axis = "T";

    double time_bnds(time=8760, bnds=2);

    float lon(rlat=680, rlon=535);
      :standard_name = "longitude";
      :long_name = "longitude";
      :units = "degrees_east";
      :_CoordinateAxisType = "Lon";

    float lat(rlat=680, rlon=535);
      :standard_name = "latitude";
      :long_name = "latitude";
      :units = "degrees_north";
      :_CoordinateAxisType = "Lat";

    double rlon(rlon=535);
      :standard_name = "grid_longitude";
      :long_name = "rotated longitude";
      :units = "degrees";
      :axis = "X";

    double rlat(rlat=680);
      :standard_name = "grid_latitude";
      :long_name = "rotated latitude";
```

```
      :units = "degrees";
      :axis = "Y";

    int rotated_pole;
      :long_name = "coordinates of the rotated North Pole";
      :grid_mapping_name = "rotated_latitude_longitude";
      :grid_north_pole_latitude = 47.0f; // float
      :grid_north_pole_longitude = -168.0f; // float

    float height_2m;
      :standard_name = "height";
      :long_name = "height above the surface";
      :units = "m";
      :positive = "up";
      :axis = "Z";

    float T_2M(time=8760, rlat=680, rlon=535);
      :standard_name = "air_temperature";
      :long_name = "2m temperature";
      :units = "K";
      :grid_mapping = "rotated_pole";
      :coordinates = "height_2m lat lon";

  // global attributes:
  :CDI = "Climate Data Interface version 1.9.8 (https://mpimet.mpg.de/cdi)";
  :Conventions = "CF-1.4";
  :history = "Mon Mar 22 18:04:48 2021: cdo selyear,1989 ../pronti/T_2M_HLDRea_002_1hr_1989.nc
T_2M_HLDRea_002_1hr_1989.nc\npost-processing";
  :source = "COSMO";
  :institution = "Fondazione CMCC (Euro-Mediterranean Center on Climate Change) - REMHI Division -
Caserta - Italy)";
  :title = "VHR-REA CCLM downscaling ERA5 (0.02 Deg)";
  :project_id = "HIGHLANDER";
  :experiment_id = "Evaluation with Urban Parametrization, DT=20sec";
  :conventionsURL = "http://www.cfconventions.org/";
  :contact = "Mario Raffa (mario.raffa@cmcc.it)";
  :references = "http//www.clm-community.eu, http://www.cmcc.it";
  :creation_date = "2021-03";
  :CDO = "Climate Data Operators version 1.9.8 (https://mpimet.mpg.de/cdo)";
}
```

3.2 Convention and Standards

In some cases, the file's contents are written using a standard netCDF convention which ensures users and automatic software that certain 'rules' have been followed when creating the file. The two most commonly used netCDF conventions for climate data are the *COARDS* and CF. The latter was created in 2003 to address the evolution of models and data sets and is a superset of the COARDS convention.

CF is evolving and has different version, for example the Copernicus Climate Data Store accepts versions not before v1.4. A netCDF file can be checked for CF compliance by using this cf-checker. In HIGHLANDER, exceptions under CF are however considered acceptable if compliant with the UNIDATA Common Data Model (CDM) to codify the Coordinate Reference System. This is the case of the **`_CoordinateAxisType`** attribute of the **Lat** and **Lon** variables in the file automatically generated by HIGHLANDER climate model runs, which does not invalidate the correctness of the format.

3.3 Tips and insights

The variables included in the downscaled climate data refers to different "heights" according to their definition and typical meaning in physical world and/or scientific

literature. Some variables (e.g. precipitation) are related to the surface, others to 2 or 10 m above ground (like temperatures or wind, respectively), others to multiple subsoil depths (soil moisture). The NetCDF format allows to handle different spatio-temporal dimensions: latitude, longitude and vertical (over surface and sub-surface) levels.

According to the Downstream Application and pre-Operational Service (DApOS) of interest, the model outputs will be further elaborated e.g. aggregating time series (from hourly to daily or monthly) and clipping the interested point(s) or area(s). Another useful transformation could be the regridding of the rlat-rlon rotated pole grid to a lat-lon grid. In rotated pole grid (as the one of HIGHLANDER climate model outputs) latitude (lat) and longitude (lon) are two-dimensions variables and rlat-rlon (rotated pole coordinates) as one-dimension variables, acting as dimensions of lat and lon. Different algorithms can be selected to regrid, according to the variable of interest, and will be implemented in the post-processing workflow through WP6.

4 Satellite data harmonization

Satellite data can sense the whole Earth surface repeatedly, allowing the mapping along time of multiple properties that characterize the specific sensed surfaces. The satellite missions are run by different space agencies, like the Unites States National Aeronautics and Space Administration (NASA); the European Space Agency (ESA); the Japanese Japan Aerospace Exploration Agency (JAXA), among others. Each space agency has different on-going missions, based on different sensing objectives, and targeting different Earth properties. Accordingly, different sensors are employed on satellites, and for each differ harmonization requirements apply.

At the present stage, Highlander is considering the use of the ESA Copernicus Sentinel 2 and possibly Sentinel 1 missions' datasets for its project purposes, and thus the focus of this chapter is on the requirements for these datasets. If other data sources will be employed in the futures, the requirements will be updated accordingly.

4.1 Copernicus INSPIRE

The INSPIRE directive aims at unifying and harmonizing cartographic information making geospatial information easily searchable, accessible, and interoperable. This is reached creating an EU spatial data infrastructure – a shared framework of standards, metadata, and tools. INSPIRE will enable the sharing of environmental geospatial information -such as satellite data and derived Earth properties- among public sector organizations, facilitating public access to spatial information across Europe. Having access to comparable information inside and across Member States is valuable for a range of applications, including carrying out environmental impact assessments, coordinating responses to natural or man-made crises, and even keeping our satellite navigation systems up-to-date.

The INSPIRE Directive is being implemented in a phased manner, with full EU-wide implementation of all provisions targeted by the end of 2021.

Copernicus, the Europe's Earth Observation program, relies heavily on accurate, up-to-date and harmonized geospatial information for the production and validation of many of its information products and services. The timely implementation of the INSPIRE directive should increase the number of available datasets and services relevant to the Copernicus Services, and considerably facilitate data discovery and access operations.

The INSPIRE Directive requires that common Implementing Rules (IR) be adopted in several specific areas (Metadata, Data Specifications, Network Services, Data and Service Sharing and Monitoring and Reporting). A user must be able to find spatial data sets and services and to establish whether they may be used and for what purpose; thus, Member States should provide descriptions in the form of metadata, compatible and usable in a Community and trans-boundary context. The INSPIRE Metadata regulation defines a set of metadata necessary to allow identification of the information resource for which metadata is created, its classification and identification of its geographic location and temporal reference, quality and validity, conformity with implementing rules on the interoperability of spatial data sets and services, constraints related to access and use, and organization responsible for the resource. Metadata elements related to the metadata record itself are also necessary to monitor that the metadata created are kept up to date, and for identifying the organization responsible for the creation and maintenance of the metadata.

As part of the Global Monitoring for Environment and Security program, the Sentinel missions support and use INSPIRE Metadata regulation. An XML INSPIRE file, including the set of metadata characterizing the User Product, is included in the product itself. Full specification regarding INSPIRE metadata and standards are provided in the Sentinel-2 and Sentinel-1 Products Specification Document.

4.2 Harmonization among different missions/sensors

For Highlander main project objectives, the ESA Sentinel 2 mission datasets will be employed. In addition, the potential use of ESA Sentinel 1 data will be also considered in a later stage, thanks to its weather independent capabilities.

SENTINEL-2 is a European wide-swath, high-resolution, multi-spectral imaging mission. The full mission specification of the twin satellites flying in the same orbit but phased at 180°, is designed to give a high revisit frequency of 5 days at the Equator.

SENTINEL-2 carries an optical instrument payload that samples 13 spectral bands: four bands at 10 m, six bands at 20 m and three bands at 60 m spatial resolution. The orbital swath width is 290 km. Each of the satellites in the SENTINEL-2 mission carries a single payload: the Multi-Spectral Instrument (MSI).

The Sentinel-1 mission comprises a constellation of two polar-orbiting satellites, operating day and night performing C-band synthetic aperture radar imaging, enabling them to acquire imagery regardless of the weather. Sentinel-1 will work in a pre-programmed operation mode to avoid conflicts and to produce a consistent long-term data archive built for applications based on long time series. The constellation will cover the entire world's land masses on a bi-

weekly basis. The Interferometric Wide Swath product will be considered for Highlander project, thanks to its double polarization (VV and VH), high spatial resolution (10m), and wide Earth coverage.

The Sentinel 1 and 2 bring information from different spectral regions, from visible to microwave. While Sentinel 2 data offers a view on the reflectance properties of vegetation or target features, Sentinel 1 report volumetric and surface scattering mechanisms, together with information regarding water content of the target.

Data integration can be beneficial for multiple analysis, and it is usually performed in machine learning -based models – foreseen in Highlander - using advanced algorithms able to exploit such different information content.

4.3 Data harmonization and quality check

With respect to quality check, Copernicus data are already provided with information on their quality for each scene, and no additional action is requested in this sense.

Harmonization of time series is often requested to derive reliable information collected in multiple dates.

For Sentinel 1, a multitemporal calibration is foreseen in preprocessing steps, together with the normalization with respect to local incidence angle (Gamma 0) for vegetation analysis. In addition, layover and shadow masks are produced during preprocessing, to exclude SAR distortion effects on slopes.

For Sentinel 2, the use of vegetation indices derived from different bands allows to normalize the signal collected from different dates; the vegetation indices use is therefore to be preferred in modeling when using inputs from time series. In addition, haze and cloud removal, or the production of cloud-free multitemporal composites, are procedures that allow to exclude invalid data caused by weather artifacts.

4.4 Harmonization with other non-satellite data sources

When additional remote sensing data are available, they could be used for specific applications developed by Highlander. This is the case of airborne data, such as lidar or hyperspectral. To be able to safely join these non-satellite inputs, routines to ensure data comparability must be set up. These include the radiometric evaluation, to check that reflectance values are comparable; also, geolocation accuracy must be evaluated, and coregistration must be performed if needed.