An Environmental Sensor Data Suite Using the OGC SensorThings API

Hylke van Der Schaaf, Jürgen Mossgraber, Sylvain Grellet, Mickaël Beaufils, Kathi Schleidt, Thomas Usländer

An environmental sensor data suite using the OGC SensorThings API

Hylke van der Schaaf[1], Jürgen Moßgraber[1], Sylvain Grellet[2], Mickaël Beaufils[2], Kathi Schleidt[3], and Thomas Usländer[1]

[1] Fraunhofer IOSB, Fraunhoferstr. 1, 76131 Karlsruhe, Germany
hylke.vanderschaaf@iosb.fraunhofer.de
juergen.mossgraber@iosb.fraunhofer.de
thomas.uslaender@iosb.fraunhofer.de
[2] BRGM, 3 avenue Claude-Guillemin, BP 36009 45060 Orléans Cedex 2, France
s.grellet@brgm.fr
m.beaufils@brgm.fr
[3] DataCove e.U., Robert Hamerling Gasse 1/14, 1150 Vienna, Austria
kathi@datacove.eu

Abstract. In many application domains sensor data contributes an important part to the situation awareness required for decision making. Examples range from environmental and climate change situations to industrial production processes. All these fields need to aggregate and fuse many data sources, the semantics of the data needs to be understood and the results must be presented to the decision makers in an accessible way. This process is already defined as the "sensor to decision chain" [11] but which solutions and technologies can be proposed for implementing it?

Since the Internet of Things (IoT) is rapidly growing with an estimated number of 30 billion sensors in 2020, it offers excellent potential to collect time-series data for improving situational awareness. The IoT brings several 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 (STA) standard, an open, unified way to interconnect devices throughout the IoT. Since its introduction in 2016, it has shown to be a versatile and easy to use standard for exchanging and managing sensor data.

This paper proposes the STA as the central part for implementing the sensor to decision chain. Furthermore, it describes several projects that successfully implemented the architecture and identifies open issues with the SensorThings API that, if solved, would further improve the usability of the API.

Keywords: open geospatial standards · sensor data management · architecture.

1 What is the OGC SensorThings API

The OGC SensorThings API [15, 9] is a standard for exchanging sensor data and metadata. It is, in many ways, a redesign of the older OGC Sensor Web Enablement (SWE) standards, targeted at the Internet of Things (IoT). The OGC SensorThings API (STA) currently consists of two parts: "part 1: Sensing" and "part 2: Tasking". The Sensing part can be seen as a successor to the OGC Sensor Observation Service, while the Tasking part overlaps specifically with the Sensor Planning Service. One of the goals of the development of the STA was to make a service that is lighter weight and easier to use than the SWE standards based on SOAP/XML that already existed, by moving to a REST approach as architectural style (see below). The API adapts the OASIS OData standard for this REST interface, offering simple data navigation, powerful filtering and the ability to customise the results to minimise the amount of data transferred and the number of necessary requests. Furthermore, the API supports push-messages to notify clients of changes in the data.

The API specifies not only the methods used to interact (Create, Read, Update, Delete) with the data in the service but also the data model that describes the data that is stored in the service.

1.1 REST as architectural style

Service-orientation has been a best-practices approach in industrial software engineering for many years. However, several architectural styles addressing how to realize the service paradigm on the concrete technological level are competing [20]. Some years ago, service orientation focused upon Web services that follow the classical architectural style of remote invocation of operations with arbitrary semantics in its behaviour, however, sometimes only specified to a certain degree leading to difficulties in semantic interoperability. On the other hand, there are the so-called RESTful Web services [16] that rely upon uniquely identifiable resources with a limited set of well-defined operations (e.g. get, set, create, delete), following the Representational State Transfer (REST) architectural style of Fielding [4].

Within the OGC SWE domain several architectural styles are co-existing. Originally starting with Web services, see the Sensor Observation Service and the Sensor Planning Service as stated above, the current development of the STA is focussing on the REST-based architectural style.

1.2 Data Model

The data model of the STA is depicted in figure 1. It is based on the ISO Observations and Measurements (O&M) conceptual model [14, 7] and consists of 8 entity types, their properties, and the relations between the entity types.

The following entity types are defined in the API:

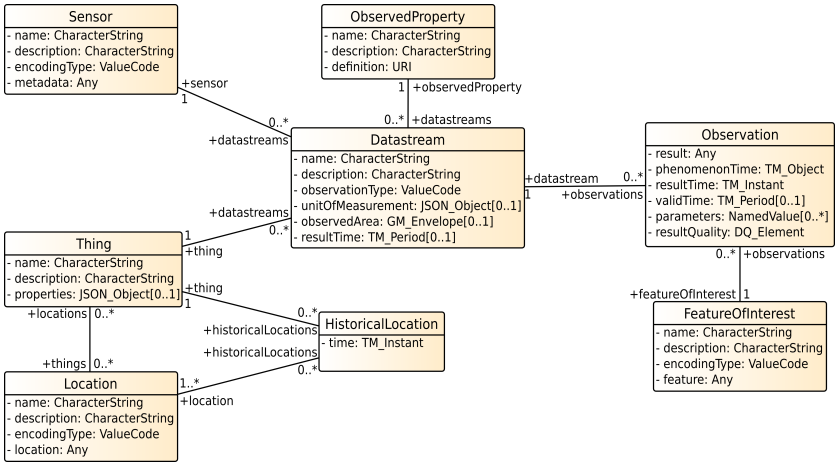

Fig. 1. The data model of the OGC SensorThings API part 1 – Sensing.

Thing What a thing is, depends completely on the individual use case. In the case of environmental monitoring, the environmental monitoring station may be a Thing, but a river, river section or ship can also be defined as a Thing.

Location The location defines where the Thing is. It can hold a machine-readable GeoJSON object describing this location, or a human-readable description, like an address.

HistoricalLocation In some cases, Things move. Every time the Location of a Thing changes, a new HistoricalLocation is created, that stores when the Thing was at the given Location.

Sensor The sensor holds the description of the device or procedure that created an Observation. This can be a relatively simple electronic device, or a complicated procedure, executed by a technician in a laboratory.

ObservedProperty The Observed Property describes the property that is being observed, like “temperature”, “humidity” or “traffic density”

Datastream A Datastream connects a set of Observations, of the same ObservedProperty, made by same Sensor, to a Thing. The Datastream also holds the unit of measurement of the Observations.

Observation The Observation object holds the actual result value of the measurement made by the Sensor. This result can be a numeric value, but can also be any other valid JSON type, including a complete JSON Object or Array.

FeatureOfInterest The FeatureOfInterest gives the Observation a location. In many cases, this will be the same as the Location of the Thing that is linked to the Observation through the Datastream, but in the case of a moving Thing, or in the case of remote sensing, the FeatureOfInterest can be different from the Location of the Thing.

Each of these entity types have several entity properties that describe exactly what data can be stored in each entity. Most of these properties are self explanatory, like name or description. Some noteworthy properties are:

Observation/phenomenonTime The phenomenonTime is the time at which the phenomenon that was observed existed. In many cases, with in-situ sensors, this is the time at which the measurement was made. In the case of ex-situ measurements, where a sample is taken that is sent to a laboratory to be analysed, it is the time when the sample was taken. In the case of predictions, the phenomenonTime can be in the future. Furthermore, the phenomenonTime can be either a time instant, or a time interval. In the case that the observation is an average over a certain time interval, the phenomenonTime can exactly reflect this time interval over which the average was taken.

Observation/result The result can be a numeric value, but can also be any other valid JSON type, including a complete JSON Object or Array.

Thing/properties The "properties" property of Thing and the parameters property of "Observation" are of the type JSON Object. The user is free to store any data in these fields. Depending on the server implementation, these fields are also searchable.

Besides entity properties, entity types also have relations, called navigation properties. Most relations are one-to-many, like the Thing-Datastream relation, and the Datastream-Observation relation. This means a Datastream has exactly one Thing, but a Thing can have zero to many Datastreams. On the "one" side of the relation, the relation is mandatory, meaning that a Datastream must have a Thing, a Sensor and an ObservedProperty and the service must ensure this requirement is met. There is one many-to-many relation in the base specification: Thing-Location. A thing can be linked to zero-to-many Locations, and each Location can be linked to zero-to-many things.

1.3 HTTP (REST) Interface

The main interface for the SensorThings API is the HTTP interface. It is based on the OASIS OData interface, but does not follow the OData specification completely. The base URL of a SensorThings API service always ends in the version number of the specification that the service implements. This makes it clear for both client and server which version of the specification is used. The 8 entity types described above each have an entity collection through which these entities can be accessed. Fetching the base URL of the API with a HTTP Get returns an index document listing the URL of each of the available entity collections.

A HTTP-Get request on a collection returns a list of the entities in the collection. Each entity can also be fetched individually by appending the entity ID, in parentheses, to the entity collection. Besides its specific properties, each entity has an ID, listed in JSON as "@iot.id", a self link, under "@iot.selfLink"

and links to other entities as described in the data model. For example, a Thing can have relations to multiple Datastreams. Therefore, each Thing has a navigation link to a collection of Datastreams, listed in its JSON under "Datastreams@iot.navigationLink", to .../v1.0/Things(*id*)/Datastreams. Likewise, each Datastream is linked to exactly one Thing. Therefore each Datastream has a navigation link to this Thing, listed under "Thing@iot.navigationLink", to .../v1.0/Datastreams(*id*)/Thing.

A request to a collection is subject to pagination, based on the request parameters and server settings. A client can request the number of entities returned to be limited using the "$top" query parameter. The server will not return more than this number of entities, but it can return fewer if it is configured with a lower maximum. If there are more entities to be returned than allowed in a single request, the server adds a link to the result, named "@iot.nextLink", that returns the next batch of entities. The client itself can also request a number of entities to be skipped, using the query parameter "$skip". For example, a client can request entities 11 to 15 using $top=5&$skip=10.

Entities in a collection can be ordered using the "$orderby" query parameter. The entities can be ordered by one or more of their properties, in ascending or descending order.

If a client is not interested in all properties of the requested entities, it can limit which properties are returned, using the query parameter "$select". For example, the query to .../v1.0/Things?$select=id,name returns only the id and the name of all Things.

When requesting entities from a collection, a filter can be applied to the entities with the "$filter" query parameter. This filter can act on any of the properties of the entities in the collection, or any of the properties of related entities. It is, for example, possible to request all Observations that have a phenomenonTime in a certain time range, or all Observations that have a Datastream that has an ObservedProperty with a certain name. The filtering options are quite extensive and include geospatial, mathematical and string functions. Multiple filters can be combined with "and", "or", and "not" keywords and parenthesis.

When requesting entities, it is possible to have related entities be directly included in the response, by using the "$expand" query parameter. The expanded items can be subjected to all query parameters, including the "$expand" query parameter itself. This makes it possible to request, in a single request, a Thing, including its ObservedProperty and Datastreams, and the latest Observation for each of these Datastreams.

Besides requesting data with HTTP-Get, new entities can be created with a HTTP-Post to the collection of the entity, entities can be updated with a HTTP-Patch or HTTP-Put to the self-link of the entity, or entities can be deleted with a HTTP-Delete to the self-link of the entity.

1.4 MQTT Interface

Besides the main HTTP interface, the SensorThings API specifies a MQTT interface as an optional extension. This MQTT interface can be used to receive

push messages when a new entity is created or when an entity is updated. The MQTT topics used to distribute these messages are the same as the URL patterns for fetching data using the HTTP interface. A subscription on the topic "v1.0/Things" receives a message whenever a Thing is created or updated. The topic "v1.0/Things(*id*)" receives a message when this specific Thing is updated and "v1.0/Datastreams(*id*)/Observations" receives a message when an Observation is added to, or updated in the given Datastream. The $select option can also be used to only receive the specified properties of the created or updated entities.

The MQTT interface can also be used to create new Observations, by sending a message to the topic "v1.0/Observations", "v1.0/Datastreams(*id*)/Observations" or "v1.0/FeaturesOfInterest(*id*)/Observations".

1.5 Extensions

The SensorThings API defines several other extensions next to the MQTT extension. The MultiDatastream extension defines a new entity type: MultiDatastream. A MultiDatastream is very similar to a normal Datastream, but instead of only one ObservedProperty and UnitOfMeasurement, a MultiDatastream can have more than one ObservedProperty, and the same number of UnitsOfMeasurement. An Observation linked to a MultiDatastream has an array as result value, that holds the same number of elements as the MultiDatastream has observed properties. MultiDatastreams are very useful for Observations of related observed properties that are always measured and used together, like wind speed and direction, or aggregate values like an average, minimum, maximum and standard deviation.

The DataArray extension defines a more efficient encoding when fetching or posting a large number of Observations. Instead of each Observation consisting of a JSON Object with name-value pairs for each property, the returned document contains a header that lists which properties are present for each Observation in the document, and the values of these properties are encoded in a single nested array.

An issue with version 1.0 of the SensorThings API is that it is not possible for a client to get a list of the extensions that a server implements. For some extensions, like the MultiDatastream extension, this is not a problem, since the presence of the extension can be deduced from the responses of the server, but for other extensions this is a problem. It is for example impossible for a client to find out if a service implements the MQTT extension, and if it does, how to connect to the MQTT service. The upcoming version 1.1 of the specification intends to solve this problem by adding the list of extensions that a service implements to the landing page, and allowing each extension to add additional information as well.

2 Architecture: From Sensor to Display

The basis for any decision is awareness of the current situation, how the current situation came to be, and how the situation is likely to change in the future. In many application domains, data from sensors and forecasting algorithms contribute an important part to this situational awareness and thus an architecture is needed to deal with the broad variety of sensors and algorithms that supply data that needs to reach the decision makers' displays [11]. This architecture needs to deal with the ingestion of sensor data, the processing and analysis of this data, and searching and display of the data. By basing the communication in the architecture on open standards, it becomes easier to mix and match data sources and potentially more information becomes available to the decision maker.

The OGC SensorThings API defines an interface for creating, updating, deleting, reading and searching data, with a versatile data model with custom data fields, and push messaging. This makes the SensorThings API suitable as a central component in a sensor data management architecture (see Fig. 2).

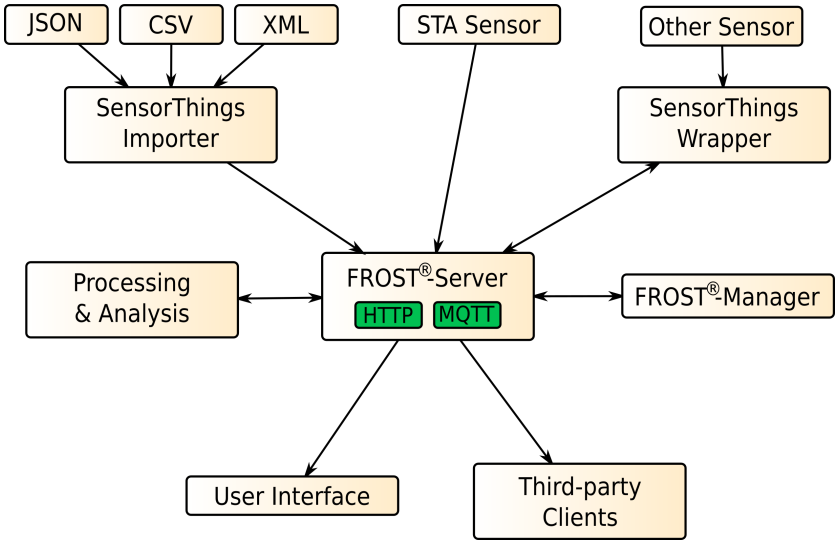

Fig. 2. An architecture for a sensor data management suite.

2.1 Components

The central component is a server that implements the SensorThings API. Any compliant server implementation that supports the features and extensions required for the use case can fill this role. An implementation, which is available

as open-source is FROST® [17]. It provides a fully standard compliant and certified implementation of the entire specification, including all extensions. It uses PostgreSQL with the PostGIS extension as data store.

In an ideal world, all sensors would be able to directly communicate using the SensorThings API. However, in the real world sensors mostly do not directly speak STA. Often sensor data needs to be imported from separate data files in non-standardised formats like CSV. To convert between the different formats, additional tools are required. Fortunately, since all that is needed to insert an Observation into a SensorThings API conformant service is an HTTP Post with a simple JSON payload, any scripting language can be used to write such importers. In more complex situations, when connecting to a data source that supplies data for multiple different sensors, like a LoRa network server, the unique identifier of the sensor in the external system can be stored in the metadata of the Sensor or the Datastream in the SensorThings server. That way the wrapper can find the matching entities in the SensorThings server with a simple search.

Processing and Analysis is an important part of sensor data management. For instance, when dealing with long time-series, statistics like hourly, daily or yearly average, minimum, maximum and standard-deviation are required to efficiently handle the data. These statistical values can be seen as (virtual) sensor data and stored in the SensorThings service together with the source data. A client that wants to visualise sensor data over a longer time frame can check which aggregation levels are available, and choose the appropriate source. This use-case of visualising aggregate values is also a good example of where the MultiDatastream extension is very useful. By using a MultiDatastream, the average, minimum, maximum and standard-deviation for an aggregation period can all be stored together, and a client does not have to fetch these values separately and match them together. The feature that the phenomenonTime of an Observation can be a time interval means there is no ambiguity as to the time interval over which the aggregate is calculated, since the start and end times of this interval can be explicitly stated. The processing algorithm can monitor the relevant Datastreams using the MQTT interface of the SensorThings API, so it is notified as soon as a new Observation is entered into the service. Since processing results are entered back into the service, they can be used as input for other algorithms. As a result, one new Observation can trigger a chain of processing runs.

While scripting is the preferred way to deal with automatic data flow, often manual access to the data is required to make small changes or correct small errors. FROST®-Manager is a GUI for interacting with SensorThings API servers. It can browse though the server and is very useful for making small updates to the metadata of entities.

Users will access the data through either the "official" user interface, usually a website of the data provider, or by using their own software. A big advantage that the SensorThings API brings for clients is the powerful query and search functionality. There is no need for a client to "download all data" just to be able to find the bits that the client really needs. Finding out what kind of data is

served by an STA instance is also trivial. In most cases a simple web-browser is sufficient, since all entities can be reached by simply following the navigation links that the API offers.

2.2 Added properties

The Thing entity type has a property called "properties" of type JSON Object, that allows users to store any meta data in a structured way. Because this is a structured data type, unlike for example the description text field, a server implementation can support direct querying on this data. If a service holds Things that are rivers or stations, the type of each Thing can be stored in "properties/type". When a client is interested in the Things of type "station", it can do the following request to only get the Things of this type:

. . . /v1.0/Things?$filter=properties/type eq 'station'

One issue with version 1.0 of the SensorThings API is that only Things and Observations have such a property for storing structured custom data. In many cases, the ability to store structured custom data on other entity types is also needed. For instance, when calculating hourly and daily aggregate values for a Datastream, it is important to be able to store the source Datastream of the aggregate data, and the aggregation level. Having this data in a structured way allows clients to better search for all aggregate MultiDatastreams for a given Datastream. To solve this deficiency, FROST® has implemented a custom extension that adds such "properties" fields to the other entity types.

Given the proven usefulness of these "properties" fields, they are also scheduled to be added to the upcoming version 1.1 of the SensorThings API.

3 Projects

In the following, first a project is presented from the environmental domain, which already successfully applied the proposed technologies. After that, two European Horizon 2020 projects are presented, which used FROST® in the domains of climate change, crisis management and cultural heritage. To widen the perspective the applicability to the INSPIRE Directive is discussed as well as an example from the industrial domain.

3.1 BRGM and French water information system Hub'Eau

The French water information system monitors the quantity and quality of all surface and ground water bodies in France. This includes data like surface and ground water levels, water flow through rivers and concentrations of over 1500 different chemical substances. These data can be a great asset for policy and decision makers, if it can be accessed in an easy and efficient way.

The French water information system architecture is traditionally based on XML webservices that serve data according to a nationally defined semantic. This architecture has been running for 20 years and the conclusion is that data

interoperability practices evolve and that the "entry ticket" to access data should be easier to maximize data reuse. Thus, an innovation project started in 2015 and an API-fication process of major national wide water related database is going on under the project 'Hub'Eau'. Swagger (OpenAPI) APIs are now progressively deployed for those. But, this solves the problem stated above only half-way as each of those API comes with its own ad-hoc API operations and semantics.

Through the French IT Research Center 'INSIDE' [5], BRGM successfully tested SensorThings API as an alternative to those home-specified APIs on the following topics (each time on national scale data): surface water levels, raw ground water levels and surface water quality. Each of those exercise proved that

- the semantic needs required by domain expert can be covered using SensorThings API;
- the required operations to search, find, reuse data are covered in SensorThings API. Actually SensorThings API goes beyond the needs expressed when it comes down to filtering and traversing the data graph;
- performance expectations for production environment are covered by FROST® (the SensorThings API implementation identified during the tests). The surface water quality database contains more than 130 Millions observations and filtering through all of them is performant and allows the deployment of GUIs on top.

As a result, those experimentations are progressively moved to production with a target to replace the home-specified APIs.

BRGM is also deploying the SensorThings API as the reference API to serve its raw groundwater levels in near real time using the API both in Write (REST, MQTT to push observations from field sensors) and Read modes. Another side effect of all those tests is that the French national database on polluted soils observations is also being exposed using SensorThings API.

3.2 beAWARE

The main goal of the Horizon 2020 project beAWARE [1] (Enhancing decision support and management services in extreme weather climate events) is to provide decision support in all phases of an emergency incident. It proposes an integrated solution that includes early warning, forecasting and analysis of multi-modal data, including sensor data. The sensor data management part of the project uses an architecture [11] as described here, centered around the OGC SensorThings API. Data is imported from a variety of external systems, using a variety of interfaces and data formats. Processing algorithms are triggered either over the MQTT interface, by subscribing to the relevant entity changes, or periodically, with result either flowing back into the SensorThings service, or pushed to other components in the system.

One of the issues identified is that support for filtering of time intervals is limited. For example, take a set of observations, where the phenomenonTime of

each observation is a time interval. When drawing a diagram of a time-based subset of these observations, one would expect to see all observations that have a phenomenonTime that overlaps with the displayed time interval. However, when requesting all observations greater than the start of the interval, and smaller than the end of the interval, this leaves out those observations that start before the displayed interval, but end in it, and those observations that start in the displayed interval, but end after it. An "overlaps" filter function does not exist for time, but would be very useful in this situation. Adding more filter functions covering Allen's interval algebra [3] would be a good addition to the standard.

3.3 HERACLES

Environmental factors, worsened by the increasing climate change impact, represent significant threats to European Cultural Heritage (CH) assets. In Europe, the huge number and diversity of CH assets, together with the different climatological sub-regions as well as the different adaptation policies to Climate Change adopted by the different Nations, generates a very complex scenario. The Horizon 2020 project HERACLES [2] has the objective to design responsive solutions for effective resilience of CH against climate change effects. Part of the developed solution is an ICT platform [12, 6] able to collect and integrate multi-source information, to provide situational awareness and decision support. Examples of relevant sensor data flowing into the platform are measurements of the environmental factors monitoring data of existing damage to CH sites and climate forecasts. Together, these data can be used analyse how climate change may impact CH sites and which measure are needed to minimise damage.

One such set of monitoring data comes from a set of accelerometers, measuring vibrations in the "Palazzo dei Consoli" in the town of Gubbio, at a data rate of 100 Hz. This large data volume is dealt with by storing time-series of half an hour per Observation, with the exact begin and end time of the series stored as time interval in the phenomenonTime, instead of creating a new Observation for each data point. This is possible in this use-case, since the sensor has a fixed measuring frequency, so the exact measuring time of each data point can be calculated from the time series. By calculating aggregate values for each minute, hour and day, the data can still be efficiently handled by clients.

3.4 INSPIRE

While less of a project in its own right, the work towards integrating SensorThings API within INSPIRE could be seen as the mother of many future projects. The European Union INSPIRE Directive laid down the foundation of a pan-European Spatial Data Infrastructure (SDI) where thousands of public sector data providers make their data, including sensor observations, available for cross-border and cross-domain reuse. This massive data pool has the potential of becoming a key source of information for decision makers in many domains, but this will only come to fruition if the data is easily findable and accessible. Current guidance documents recommend the provision of this data

via OGC services, foreseeing utilization of WMS, WFS, SOS and WCS for data provision. As technology and standards have progressed, the various additional technical requirements have been raised by SDI stakeholders, foremost:

- the need for adoption of RESTful architectures
- alternative (to GML) data encodings, such as JavaScript Object Notation (JSON) and binary exchange formats
- adoption of asynchronous publish–subscribe-based messaging protocols.

This inspired us to explore the suitability of SensorThings API for fulfilling the requirements laid down by the INSPIRE Directive pertaining to:

- Data Scope: Within INSPIRE, data models have been defined encompassing all attributes of a spatial feature deemed relevant within that spatial data theme. The challenge lay in mapping existing SensorThings API attributes to those foreseen within the INSPIRE themes. For attributes required by INSPIRE where a suitable mapping could not be made, additional attributes within the extended properties section foreseen within the 1.1 version of SensorThings API where defined [8].
- Download Service Requirements: INSPIRE lists various functionalities required by network services within COMMISSION REGULATION (EC) No 976/2009 of 19 October 2009 implementing Directive 2007/2/EC of the European Parliament and of the Council as regards the Network Services (OJ L 274, 20.10.2009, p. 9) [19]. At present, work is progressing on showing how all these requirements can be fulfilled either by SensorThings API as it currently stands, or via extensions planned for the upcoming 1.1 version of the standard [18].

At present, an INSPIRE Good Practice is being set up to finalize this work, and illustrate how SensorThings API can be utilized to fulfill INSPIRE requirements while providing the available data in a simple and easy to use manner.

3.5 AutoInspect

Although the main application domain of the STA is environmental information and risk management systems, the STA may also be applied in other domains of the Internet of Things (IoT). The industrial branch of the IoT, the Industrial IoT (IIoT), comprises application domains such as energy management, logistics or industrial production. For example, manufacturing plants encompass quality inspection activities, sometimes offline in dedicated inspection stations, but more and more inline, e.g. camera-based systems that try to detect defects in body parts of automobiles. Such inspection activities may be modelled by means of the STA-based sensor data model. Their results (e.g. images or roughness maps) are observations of these inspection tasks. The benefit of applying the STA is to have a common interface and data model for all kind of inspection activities which reduces the engineering effort to feed the individual inspection results

into overarching quality management systems of a production plant. The STA perfectly complements other standards that are widespread in automation technology such as IEC 62541 OPC UA (Open Platform Communications Unified Architecture) of the OPC Foundation [13]. Fraunhofer IOSB currently experiments with both standards, the STA and OPC UA together with the open source OPC UA implementation hosted at https://open62541.org/, in its automotive quality inspection pilot site entitled AutoInspect. Here, OPC UA is used for controlling and managing the individual inspection tasks, whereby the SensorThings API is used to store and retrieve the inspection results in raw and aggregated or interpreted form.

4 Conclusions

The OGC SensorThings API already proved its value as the central part of implementing the sensor to decision chain in several scientific as well as real world projects. It can be facilitated to implement the storage of heterogeneous data, attach semantic annotations, as well as trigger processing algorithms with the new Tasking extension of the standard.

It supports advanced filtering, not just on the properties of the entities being queried, but also across the relations of those entities, including the structured, custom meta-data. The MQTT extension allows clients to receive push notifications when entities are added or changed. All this makes the OGC SensorThings API a good core in a sensor data management architecture as demonstrated here. One improvement that would greatly increase the functionality of the API is adding a properties field to all other entity types besides the Thing entity type. Fortunately, this change is coming in the upcoming version 1.1 of the API.

A second issue that was identified is that it is not possible to see if a server implements the MQTT extension, and if it does, how to connect to this MQTT service. The upcoming version 1.1 of the standard is scheduled to solve this issue by adding a list of implemented extensions to the landing page of the service, and allowing each extension to add additional information. The MQTT extension can use this to list the connection details that a client can use to connect to the MQTT service.

The standard lacks functions for effective filtering when dealing with time intervals, which is noticeable when dealing with observations that have a “validTime”, or a “phenomenonTime” that is an interval. This lack of filter functions that support time interval logic can be addressed with an extension to the standard, that adds a set of functions covering Allen’s interval algebra.

Future work will include using the Tasking part of the SensorThings API [10] to control on-demand processing. This will allow the processing algorithms to be started that do not have a clear trigger based on incoming data, but are run on-demand based on user interaction.

Acknowledgments

The HERACLES project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700395.

The beAWARE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 700475.

Funding for BRGM work is provided by its joint IT Research Center 'INSIDE' dedicated to innovation in Environmental Information Systems with the French National Agency for Biodiversity (AFB), French Museum of Natural History (MNHN) and French Marine Agency (IFREMER)

References

1. beAWARE Project, https://beaware-project.eu/
2. HERACLES Project, http://www.heracles-project.eu/
3. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM **26**(11), 832–843 (Nov 1983). https://doi.org/10.1145/182.358434, http://doi.acm.org/10.1145/182.358434
4. Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California, Irvine (2000)
5. Grellet, S.: Implementation of ogc sensorthing api for environmental data (2018), https://github.com/INSIDE-information-systems/SensorThingsAPI
6. Hellmund, T., Hertweck, P., Hilbring, D., Mossgraber, J., Alexandrakis, G., Pouli, P., Siatou, A., Padeletti, G.: Introducing the heracles ontology—semantics for cultural heritage management. Heritage **1**(2), 377–391 (Nov 2018). https://doi.org/10.3390/heritage1020026
7. ISO/DIS: 19156:2011 geographic information — observations and measurements (2011), https://www.iso.org/standard/32574.html
8. Kotsev, A., Schleidt, K., Liang, S., van der Schaaf, H., Khalafbeigi, T., Grellet, S., Lutz, M., Jirka, S., Beaufils, M.: Extending inspire to the internet of things through sensorthings api. Geosciences **8**(6), 221 (Jun 2018). https://doi.org/10.3390/geosciences8060221, http://dx.doi.org/10.3390/geosciences8060221
9. Liang, S., Huang, C.Y., Khalafbeigi, T.: OGC SensorThings API part 1: Sensing, version 1.0, 15-078r6 (2016), http://docs.opengeospatial.org/is/15-078r6/15-078r6.html
10. Liang, S., Khalafbeigi, T.: OGC SensorThings API part 2: Tasking core, version 1.0, 17-079r1 (2016), http://docs.opengeospatial.org/is/15-078r6/15-078r6.html
11. Moßgraber, J., Hilbring, D., van der Schaaf, H., Hertweck, P., Kontopoulos, E., Mitzias, P., Karakostas, A., Vrochidis, S., Kompatsiaris, I.: The sensor to decision chain in crisis management. In: Boersman, K., Tomaszewski, B. (eds.) Conference Proceedings of the 15th International Conference on Information Systems for Crisis Response and Management. pp. 754–763 (05 2018)
12. Moßgraber, J., Lortal, G., Calabro, F., Corsi, M.: An ict platform to support decision makers with cultural heritage protection against climate events. In: Geophysical Research Abstracts. vol. 20 (04 2018), https://meetingorganizer.copernicus.org/EGU2018/EGU2018-13962.pdf

13. OPC Foundation: Unified architecture (2008), https://opcfoundation.org/about/opc-technologies/opc-ua/
14. Open Geospatial Consortium: Observations and measurements (2011), https://www.opengeospatial.org/standards/om
15. Open Geospatial Consortium: OGC SensorThings API (2016), https://www.opengeospatial.org/standards/sensorthings
16. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (2008)
17. van der Schaaf, H., Jacoby, M.: FROST-Server (2016), https://github.com/FraunhoferIOSB/FROST-Server
18. Schleidt, K.: SensorThings work at DataCove (2018), https://github.com/DataCoveEU/SensorThings
19. The Commission of the European Communities: Commission regulation (ec) no 976/2009 of 19 october 2009 implementing directive 2007/2/ec of the european parliament and of the council as regards the network services (2009), https://eur-lex.europa.eu/eli/reg/2009/976/2010-12-28
20. Usländer, T.: Service-oriented design of environmental information systems. Ph.D. thesis, Universität Karlsruhe (2010). https://doi.org/10.5445/KSP/1000016721