Open Geospatial Consortium

Submission Date: 2022-05-06

Approval Date: YYYY-MM-DD

Publication Date: YYYY-MM-DD

External identifier of this OGC® document: http://www.opengis.net/doc/bp/21-068

Internal reference number of this OGC® document: 21-068

Category: OGC® Best Practice

Editor: Andreas Matheus

OGC Best Practice for using SensorThings API with Citizen Science

Warning

This document defines an OGC Best Practice on a particular technology or approach related to an OGC standard. This document is not an OGC Standard and may not be referred to as an OGC Standard. It is subject to change without notice. However, this document is an official position of the OGC membership on this particular technology topic.

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

Document type: OGC® Best Practice
Document subtype:
Document stage: Draft
Document language: English

restriction (except as set forth below), including without limitation the rights to implement, use, copy, modify, merge, publish, distribute, and/or sublicense copies of the Intellectual Property, and to permit persons to whom the Intellectual Property is furnished to do so, provided that all copyright notices on the intellectual property are retained intact and that each person to whom the Intellectual Property is furnished agrees to the terms of this Agreement.

If you modify the Intellectual Property, all copies of the modified Intellectual Property must include, in addition to the above copyright notice, a notice that the Intellectual Property includes modifications that have not been approved or adopted by LICENSOR.

THIS LICENSE IS A COPYRIGHT LICENSE ONLY, AND DOES NOT CONVEY ANY RIGHTS UNDER ANY PATENTS THAT MAY BE IN FORCE ANYWHERE IN THE WORLD.

THE INTELLECTUAL PROPERTY IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE DO NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE INTELLECTUAL PROPERTY WILL MEET YOUR REQUIREMENTS OR THAT THE OPERATION OF THE INTELLECTUAL PROPERTY WILL BE UNINTERRUPTED OR ERROR FREE. ANY USE OF THE INTELLECTUAL PROPERTY SHALL BE MADE ENTIRELY AT THE USER'S OWN RISK. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR ANY CONTRIBUTOR OF INTELLECTUAL PROPERTY RIGHTS TO THE INTELLECTUAL PROPERTY BE LIABLE FOR ANY CLAIM, OR ANY DIRECT, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM ANY ALLEGED INFRINGEMENT OR ANY LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR UNDER ANY OTHER LEGAL THEORY, ARISING OUT OF OR IN CONNECTION WITH THE IMPLEMENTATION, USE, COMMERCIALIZATION OR PERFORMANCE OF THIS INTELLECTUAL PROPERTY.

This license is effective until terminated. You may terminate it at any time by destroying the Intellectual Property together with all copies in any form. The license will also terminate if you fail to comply with any term or condition of this Agreement. Except as provided in the following sentence, no such termination of this license shall require the termination of any third party end-user sublicense to the Intellectual Property which is in force as of the date of notice of such termination. In addition, should the Intellectual Property, or the operation of the Intellectual Property, infringe, or in LICENSOR's sole opinion be likely to infringe, any patent, copyright, trademark or other right of a third party, you agree that LICENSOR, in its sole discretion, may terminate this license without any compensation or liability to you, your licensees or any other party. You agree upon termination of any kind to destroy or cause to be destroyed the Intellectual Property together with all copies in any form, whether held by you or by any third party.

Except as contained in this notice, the name of LICENSOR or of any other holder of a copyright in all or part of the Intellectual Property shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Intellectual Property without prior written authorization of LICENSOR or such copyright holder. LICENSOR is and shall at all times be the sole entity that may authorize you or any third party to use certification marks, trademarks or other special designations to indicate compliance with any LICENSOR standards or specifications. This Agreement is governed by the laws of the Commonwealth of Massachusetts. The application to this Agreement of the United Nations Convention on Contracts for the International Sale of Goods is hereby expressly excluded. In the event any provision of this Agreement shall be deemed unenforceable, void or invalid, such provision shall be modified so as to make it valid and enforceable, and as so modified the entire Agreement shall remain in full force and effect. No decision, action or inaction by LICENSOR shall be construed to be a waiver of any rights or remedies available to it.

i. Abstract

This document introduces an extension to the OGC SensorThings data model and discusses the best practices for using such an extension in the context of Citizen Science.

The motivation for the introduced extension STAplus has been developed during the EC H2020 project Cos4Cloud and is based on requirements from Citizen Science. Whereas the dominant use of the OGC SensorThings data model (and API) can be coined with the use case "single authority provides sensor readings to consumers", in Citizen Science there are many contributors (citizens) that – together – create the big "picture" with their observations.

The introduced extension "STAplus" supports to model that those observations are owned by (different) users that may express the license for re-use; we call this part of the contribution the ownership concept. In addition to the ownership and license abilities, the introduced extension allows to express explicit relations between observations and to create group(s) of observations to containerize observations that belong together. Relations can be created among any individual observations or observations of a group to support performant Linked Data extraction and semantic queries, e.g. expressed in SPARQL.

We believe that the introduced extension is an important contribution towards the realization of the FAIR principles perhaps not only in Citizen Science as STAplus strengthens the "I" (Interoperability) through a common data model and API as well as the "R" (Re-usability) by allowing to express standards-based queries that may consider licensing conditions, relevant for reuse of other users' observations. The STAplus Data Model and Business Logic also enriches existing deployments as the extension can be seamlessly added and thereby offer new capabilities to create and manage the "big picture" with multi-user capabilities.

This document also illustrates best practices of using STAplus, evaluated with proof-of-concept deployments based on the implementations by 52°North, Secure Dimensions and CREAF.

ii. Keywords

The following are keywords to be used by search engines and document catalogues:

SensorThings, STA, API, STAplus, Extension, Citizen Science, Linked Data, Business Logic, Security Considerations

iii. Preface

This document is the result of many discussions and iterations that took place in the Cos4Cloud project with the objective to define an extension for the existing OGC SensorThings API, Part 1: Sensing Version 1.1 to improve the reusability and interoperability among Citizen Science data.

This document presents the „final" consensus of a SensorThings data model extension named STAplus as per February 2022. Different iterations of the extension evolution were presented at OGC meetings dating back to early 2020. The extension was first presented in the Citizen Science DWG, then the Architecture DWG and finally introduced to the SWE IoT SWG. The received comments and feedback were incorporated into the extension (data model and business logic) presented in this document.

iv. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium Inc.

Secure Dimensions, Germany
CREAF, Spain
52°North, Germany
Fraunhofer-Gesellschaft, Germany

v. Submitters

All questions regarding this submission should be directed to the editors or the submitters:

Name	**Affiliation**
Andreas Matheus	Secure Dimensions
Joan Maso and Kaori Otsu	CREAF
Henning Bredel	52°North
Hylke van der Schaaf	Fraunhofer-Gesellschaft

vi. Acknowledgements

The writing of this Best Practice and the supporting work presented in this document was undertaken within the Cos4Cloud project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 863463.

The computing cloud resources to undertake the proof of concept and evaluation deployment for the Secure Dimensions implementation was made available by EGI.

We like to thank Mathias Chouet from INRIA for their contributions.

We also like to thank the OGC members from the SWE.IoT working groups (DWG and SWG) for their constructive feedback.

vii. History

Date	Release	Author	Clauses	Description
17.11.2021	0.1	Andreas Matheus	All	Creation
25.11.2021	0.2	Henning Bredel	All	First feedback
28.01.2022	0.3	Kaori Otsu	5.1.2, 7.1, 7.1.2	Applying STAplus to Natusfera
03.02.2022	0.3	Hylke van der Schaaf	7.3	Leveraging Batch-Processing to improve the Camera Trap use case
14.02.2022	0.4	Andreas Matheus	All	Cleanup and re-organization of text to improve reading
08.02.2022 22.03.2022		Mathias Chouet	5.3, 7, 9.1.6, 10.5, 6.5	Creation
09.03.2022		Henning Bredel	6.7	Creation
22.03.2022		Joan Maso	6.6, 6.6.1, 7.13	Creation
04.04.2022		Andreas Matheus	6.6.2	STAplus Viewer App added
04.04.2022	0.5	Andreas Matheus	All	Accept changes
06.04.2022		Kaori Otsu	All	Collaborative editing

		Joan Maso Henning Bredel Hylke van der Schaaf Andreas Matheus		
09.04.2022		Joan Maso	6.5, 8.4	Creation
12.04.2022		Henning Bredel	3.5, 5.7	Motivation for Modelling Community Processes
22.04.2022	0.6	Kaori Otsu	All	Re-organization and editing
26.04.2022		Andreas Matheus	All	Clarifying open and completing missing items Incorporating comments
02.05.2022	0.7	Kaori Otsu	All	Final checks
04.05.2022	0.8	Andreas Matheus	All	Final edits and formatting

viii. Table of Content

I. ABSTRACT 3
II. KEYWORDS 3
III. PREFACE 3
IV. SUBMITTING ORGANIZATIONS 4
V. SUBMITTERS 4
VI. ACKNOWLEDGEMENTS 4
VII. HISTORY 4
VIII. TABLE OF CONTENT 5

1. INTRODUCTION 9
1.1 CONCEPT OF OWNERSHIP 9
1.2 IMPROVING F A I REUSABILITY 10
1.3 CREATING OBSERVATION BAGS 10
1.4 EXPRESSING RELATIONS 10
1.5 SCOPE 11

2. REFERENCES 12

3. BUSINESS VALUE 13
3.1 MOTIVATION 13
3.2 The Camera Trap Use Case 14
3.3 Biodiversity Use Case Natusfera/iNaturalist 14
3.4 Biodiversity Use Case Pl@ntNet as a Service 14
3.5 Community Process Use Case(s) 15

4. DATA MODEL 16

5. BEST PRACTICES 18
5.1 BEST PRACTICE WHEN MODELLING THE CAMERA TRAP USE CASE 18
5.1.1 Setup of the Camera Trap 19
5.1.2 Camera Trap at Runtime 23
5.2 BEST PRACTICE LEVERAGING THE STA API BATCH-PROCESSING 27
5.2.1 Camera Trap Initialization using Batch-Processing 27
5.2.2 Camera Trap Runtime using Batch Processing 32
5.3 BEST PRACTICE WHEN APPLYING STAPLUS TO NATUSFERA/INATURALIST DATA 34
5.3.1 Schema mapping from Natusfera to STAplus 36
5.3.2 Testing roles of Relation class with Darwin Core terms 39
5.4 BEST PRACTICE WHEN IMPORTING NATUSFERA/INATURALIST INTO STAPLUS 40
5.5 BEST PRACTICE WHEN MODELLING PL@NTNET AS A SERVICE 42

5.6 BEST PRACTICE ON HOW TO VISUALIZE STAPLUS DATA IN A MAP 44
5.6.1 STAplus Viewer App by CREAF *45*
5.6.2 STAplus Viewer App by Secure Dimensions *48*
5.7 BEST PRACTICE MODELLING COMMUNITY PROCESSES 53
5.7.1 A contributor uploads some observations about an occurrence of an animal *54*
5.7.2 Contributors enrich data in a particular community process *55*
5.7.3 A Scientist assembles data as a referenceable data collection *55*
5.7.4 Autonomous, user detached task regularly runs a probability check *56*
5.7.5 Summary of recommended practices *57*

6. CONSIDERATIONS ON THE BUSINESS LOGIC 58
6.1 CONCEPT OF OWNERSHIP 58
6.1.1 Illustrating the Concept of Ownership for the Camera Trap Use Case *59*
6.2 SECURITY CONSIDERATIONS 60
6.2.1 STAplus Impersonation Attacks *60*
6.2.2 Denial of Service *61*
6.3 GENERAL CONSIDERATIONS ON CREATE, UPDATE AND DELETE OPERATIONS 61
6.3.1 Create Operation *61*
6.3.2 Update Operation *62*
6.3.3 Delete Operation *62*
6.4 CONSIDERATIONS ON CLASS PARTY 63
6.4.1 Single Party Instance *63*
6.4.2 Direct Impersonation *63*
6.4.3 Indirect Impersonation *63*
6.4.4 Delete Operation *63*
6.5 CONSIDERATIONS ON CLASS DATASTREAM / MULTIDATASTREAM 64
6.6 CONSIDERATIONS ON CLASS THING 64
6.6.1 Update *65*
6.6.2 Delete *65*
6.7 CONSIDERATIONS ON CLASS SENSOR 65
6.8 CONSIDERATIONS ON CLASS FEATUREOFINTEREST 65
6.9 CONSIDERATIONS ON CLASS OBSERVEDPROPERTY 66
6.10 CONSIDERATIONS ON CLASS LICENSE 66
6.11 CONSIDERATIONS ON CLASS GROUP 66
6.12 CONSIDERATIONS ON GROUPS VERSUS MULTIDATASTREAMS 68
6.13 CONSIDERATIONS ON CLASS RELATION 68
6.14 CONSIDERATIONS ON CLASS OBSERVATION 69

7. SENSORTHINGS CONVENIENCE API 70

8. IMPLEMENTATIONS 71
8.1 FRAUNHOFER'S FROST-SERVER EXTENSION BY SECURE DIMENSIONS 71
8.2 FRAUNHOFER'S FROST-SERVER CONVENIENCE API BY SECURE DIMENSIONS 72
8.3 STAPLUS BASH CLIENT FOR CAMERA TRAP DATA UPLOAD BY SECURE DIMENSIONS 73
8.4 52°NORTH 73
8.5 STAPLUS JAVASCRIPT WEB-APP BY CREAF 73
8.6 PL@NTNET AS A SERVICE BY INRIA 73
8.7 POSTGRESQL BACKEND INDEXATION 74
8.8 SENSORTHINGS API MAP LIBRARY 75
8.9 EXCEL ODATA DATA FEED 78

9. FUTURE WORK 80
9.1 TECHNICAL ASPECTS 80
9.2 PROCEDURAL ASPECTS 80

APPENDIX A: STAPLUS DATA MODEL 81

Figures:

Figure 1: SensorThings API with Citizen Science 11
Figure 2: STAplus extension to Datastream 16
Figure 3: STAplus extension to MultiDatastream 16
Figure 4: Data captured by the Camera Trap 18
Figure 5: Mapping the trap event data to STAplus (overview) 23
Figure 6: Entities, their linking and separation into initialization and runtime 27
Figure 7: A screenshot of the web interface in Natusfera portal showing observation group ID 313411 (source: https://natusfera.gbif.es/observations/313411) 35
Figure 8: Schema mapping for Party 36
Figure 9: Schema mapping for Thing 37
Figure 10: Schema mapping for Observation 37
Figure 11: Schema mapping for FeatureOfInterest 37
Figure 12: Schema mapping for Group 38
Figure 13: Schema mapping for Datastream 38
Figure 14: Schema mapping for Project 39
Figure 15: Schema mapping for License 39
Figure 16: User interface of the Natusfera/iNaturalist to STAplus routine 41
Figure 17: Pl@ntNet observation (ID 1000000064): 42
Figure 18: STAplus response shown in the interactive MiraMon Map Browser 48
Figure 19: STAplus viewer app (https://cos4cloud.secd.eu/staplus-viewer-app) showing Pl@ntNet data 49
Figure 20: STAplus viewer app showing Pl@ntNet the result of querying by location 51
Figure 21: OpenLayers based implementation of the STAplus Viewer App (https://cos4cloud.secd.eu/staplus-viewer-app/ol.html) 53
Figure 22: STAplus Citizen Observatories use cases 54
Figure 23: STAplus Viewer App connecting to an access protected STAplus endpoint 76
Figure 24: STAplus Viewer App connecting to an access protected STAplus endpoint, displaying FeaturesOfInterest 77
Figure 25: Microsoft Office Excel version 2019 view of the STAplus data 78
Figure 26: Visualization of the Group observations for FeatureOfInterest (ID 9357) 79
Figure 27: STAplus extension to Datastream 81
Figure 28: STAplus extension to MultiDatastream 82

Tables:

Table 1: STAplus Classes and their use with Pl@ntNet 44
Table 2: STAplus query divided in parts and explained 46
Table 3: Comparing the code used in Leaflet and in OpenLayers 52
Table 4: Common STAplus queries 74

Examples:

Example 1: Request for creating a project (HTTP POST to /Projects) 19
Example 2: Request to create the Raspberry Pi thing and its datastream (HTTP POST /Things) 20
Example 3: Request to create the sensor board thing and its MultiDatastream instance (HTTP POST /Things) 21
Example 4: Fetch the datastream identified by UUID 22
Example 5: Fetch the datastream iot.id by UUID 22

Example 6: Fetch all datastream iot.id and uuid property for the thing 22
Example 7: Response of iot.id and uuid properties for datastream associated to Thing(17)... 22
Example 8: Response of the multi-datastream for the sensor board..... 23
Example 9: FeatureOfInterest instance for the animal detected at a Camera Trap event..... 24
Example 10: FeatureOfInterest instance of the sensor measurements for a Camera Trap event 24
Example 11: Observation representing the animal (part one) 25
Example 12: Example POST request using CURL to upload binary observation 25
Example 13: Generating the group (each observation has its own uuid property)..... 26
Example 14: Fetch all observation iot.id and uuid properties for the created group 26
Example 15: Response of iot.id and uuid properties for observations associated to Group(1) 26
Example 16: Creating a Darwin Core relation that relates the observed photo of the animal to the DWC identifier..... 26
Example 17: Batch-Processing initialization example for the STAplus endpoint to setup the Camera Trap (*https://gist.github.com/hylkevds/83cde8c4b8b561ffbab12bc1bb594251*)..... 31
Example 18: Batch-Processing update example for the STAplus endpoint once the Camera Trap setup has changed..... 31
Example 19: Batch-Processing runtime example for uploading Camera Trap event data to the STAplus endpoint (*https://gist.github.com/hylkevds/9b88122bedc05abfc0226427ff0d26dd*) 34
Example 20: Natusfera observation data for ID 313411 encoded in JSON format (Source: *https://natusfera.gbif.es/observations/313411.json*)..... 36
Example 21: Create a relation with dwc: recordedBy 39
Example 22: Create a relation with dwc: identifiedBy..... 40
Example 23: Create a relation with dwc: inCollection where observation is photo..... 40
Example 24: Create a relation with dwc: inCollection where observation is identification.... 40
Example 25: Create a relation with dwc: toTaxon..... 40
Example 26: Pl@ntNet native JSON format is pretty straightforward; it contains human-readable public information about the observation..... 43
Example 27: Pl@ntNet observation data mapped to STAplus format 44
Example 28: Response of the STAplus query 47
Example 29: Nested request of the STAplus query..... 47
Example 29: STAM queryObject configuration..... 49
Example 30: STAM queryObject returning count for feature-of-interest 50
Example 31: STAM markerClick example..... 50
Example 32: STAM markerClick STAplus specific query via the Group class 50
Example 33: Create thing directly via /Things path 61
Example 34: Create thing indirectly via /Datastreams path..... 62
Example 35: Example of a POST request that adds an observation “inline” 67
Example 36: Example of a POST request that adds an Observation “linked” 67
Example 37: Example CURL request to upload a binary observation 70
Example 38: SQL queries to accelerate performance of the implementation by adding indexing to the database..... 75

1. Introduction

The Citizen Science community has been collecting user contributions for years. Very many operators host portals in their domain, offering to participate and create observations in a particular project. These projects span a wide spectrum of focus: i.e. from environmental, biological to socioecological topics. All of these portal operators have their own database (data model) and API which mainly serves their purpose to commence upload of (raw) observations and illustration of the results on their own web and mobile application. This has led to solutions where Citizen Science data exists in silos. Due to the different (mostly flat) data models exposed via the APIs and the simplistic nature of the APIs, it is actually really hard to bring valuable user contributions together, if they are stored at different operators.

Trying to understand what actually must be improved, we believe that realizing the FAIR (the Findable, Accessible, Interoperable and Re-useable) principle by applying a common generic data model and API to Citizen Science data would be a big step forward. This is not extremely difficult as separate (proxy) APIs could be developed to provide an interoperable access to the observations. In that sense, this document introduces the approach of adding the OGC SensorThings API to existing Citizen Science portals to demonstrate how to improve the aspects of interoperability and re-use supporting the FAIR principle.

Within this document, we present an extension to the OGC SensorThings API named STAplus. This extension originally started with motivations and requirements from Citizen Science. However, we believe that the extension STAplus, as introduced in this document, has wider applicability than just Citizen Science.

STAplus is a 100% backwards compatible extension to SensorThings V1.1, so it can be added to any existing deployments.

1.1 Concept of Ownership

In Citizen Science, very many users participate in projects or campaigns, offered by different Citizen Science portals. These portals, typically operated by different entities have one feature in common: Contributions, uploaded by users are associated with the user and the ownership does not change. We could say that there is an explicit relationship between the observation (contribution) and the user. Expressed as ownership, users can undertake certain actions to their resources.

With the SensorThings API, observations are linked to a datastream that is linked to a sensor which belongs to a thing. The data model does not provide a class that allows to explicitly link a user (party) to an observation. This limits the use of the SensorThings data model (and API) to a simple use case: One operator can create data objects and all other users have read-only access. This limits the options for applications to interact with the API.

Even though, the V1.1 of the SensorThings data model offers the generic use of “properties”, we don’t believe it is wise to express ownership (user association) to be buried in properties. It also makes GDPR compliance difficult for applications using the API, as it is unclear if properties store personal data and therefore fall under GDPR. It decreases the interoperability tremendously, as one would need to know which attribute expresses ownership. Also, querying observations based on unstructured properties is extremely complex and difficult.

Therefore, the STAplus extension defines the class Party for expressing the association from a (Multi)Datastream to a user. All observations, generated by a Datastream instance belong to the associated party.

1.2 Improving F A I Reusability

In general, there are many aspects when it comes to ensure reusability of (existing) data. Not only in the context of Citizen Science, one fundamental aspect is the licensing aspect. Very many users contribute to Citizen Science with the motivation to do meaningful things for the common good. But at the same time, they like to be named when it comes to re-use of their contribution(s). Therefore, most contributions on Citizen Science are freely accessible (open access) but the re-use is not simply "open". In order to get credited, users might associate a license like CC-BY (Creative Commons Attribution License). Even though the data is still freely accessible, there is a condition that must be followed as expressed in the license.

1.3 Creating Observation Bags

When contributing to Citizen Science, the actual observation is often a set or bag of individual observations that belong together; belong to the same observation event. For example, a camera trap event contains of a picture, a textual observation expanding the likelihood of species prediction and sensor readings for environmental context (temperature, humidity, luminance, air pressure, GPS location). All of these (individual) observations got created at the same time/location and could therefore be grouped.

Another use case for applying grouping to existing observations is to create a package of observations for the purpose of building the fundamentals for research or to be used in workflows. For researching and later evaluation of a particular phenomenon, the same bag (or set) of observations can be exchanged via the grouping concept.

Also, over time a user community might link other observations and even provide cross links to other databases like GBIF. These can be semantically tagged with the Relation class which is described later in this document.

The STAplus extension defines a flexible grouping concept by adding the class Group to the SensorThings data model.

1.4 Expressing Relations

For Citizen Science it is important to express relations between observations explicitly to support search based on these relations. The SensorThings data model does not support to express relationships.

Therefore, the STAplus extension introduces the class Relation that allows to create generic "from – to" relations. The "to" can point to another observation or to an external object. This allows the generic expression of meaning leveraging existing semantic concepts that exist elsewhere: e.g. Dublin or Darwin Core. This helps to reason how observations are related even if they were not observed in a same observation event, but were linked later on in a particular community process (like linking to other databases).

The use of Relations as defined in STAplus can be applied to already existing SensorThings deployments to enrich the data towards semantics.

1.5 Scope

Targeting at Best Practice to use OGC Sensor Things API Part 1: Sensing Version 1.1 with Citizen Science, this document is organized in different sections, as illustrated in the following figure.

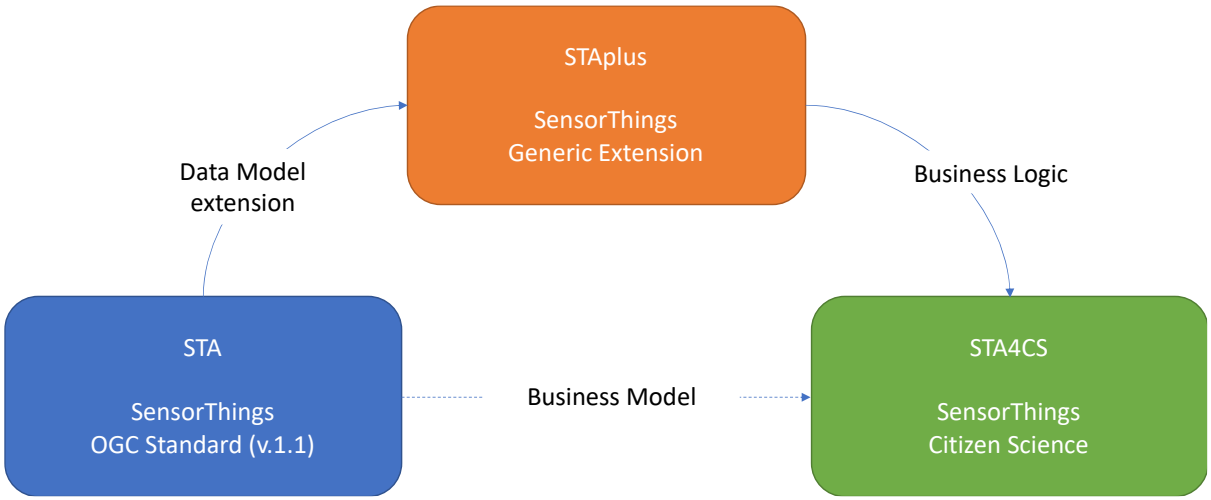

Figure 1: SensorThings API with Citizen Science

The first part of this document describes the generic extension to the SensorThings Data Model: STAplus. The resulting STAplus Data Model was designed to function in very many use cases and in particular enrich those that are native to the sensor things world; In other words, the STAplus Data Model is not limited to just be used with Citizen Science!

The extension allows expressing the following additional characteristics:

- **P**eople: The class `Party` allows to link a user to a Datastream or Group
- **L**icense: The class `License` allows to express reuse conditions by linking a License to a Datastream and / or to a Group. A License on a Datastream has the result that all Observations of that Datastream (as well as entities of the Thing, Sensor and ObservedProperty) have the associated license. A License on a Group allows to give the bag or set of Observations (represented by the Group) a license for reuse of the Group itself. It is important to note that still the license for each Observation must be followed.
- **U**nion: The class `Group` allows to package individual Observations as a bag or set either as deep copy or via linking
- **S**emantics: The class `Relation` supports to express relationships between Observations using the "from-to" type. It is also possible to create relations between Observation entities and external objects using the URI scheme. This allows in particular to express a relation to any entity of the database (via their external URI). Relations can exist on its own or be included into a Group to enrich a bag or set of Observations.
- Project: The class `Project` is a container of Datastream and MultiDatastream entities that allows to organize a campaign or project and to provide metadata as well as legal information such as terms of use and a privacy statement in case it is relevant.

The second part of this document focuses on defining a generic Business Logic that fits the STAplus extension. After outlining and discussing the Business Logic in detail, this document provides best practices how to apply the STAplus and Business Logic to Citizen Science. We do this based on use cases extracted from the Cos4Cloud project:

- Camera Trap: The Cos4Cloud partner DynAikon developed a camera trap software for the automatic recording of animals and their automated species detection. STAplus is used to illustrate how the captured information can be structured and organized to be stored and retrieved based on groups that represent camera trap events.
- Natusfera / iNaturalist: How can records from an existing Citizen Science biodiversity portal be represented using STAplus. The result is available from the STAplus deployment in the EGI (EOSC) cloud: https://cos4cloud.secd.eu/natusfera/v1.1
- Pl@ntNet: How can records from an existing Citizen Science biodiversity portal be represented using STAplus. This use case illustrates the concept of "observation bags" as well as query performance concerns with a 10M+ dataset.
- STAplus-Viewer is a simple JavaScript based Web application to illustrate how to request and render data received via the STAplus data model extension. The emphasis on this best practice is to illustrate how meaningful and powerful queries can be created, specifically on how to use the classes `Group` and `Relation`.

Towards the end of the document, we discuss relevant security considerations, summarize the results and introduce recommendations for future work.

2. References

OGC Sensor Things API Part 1: Sensing Version 1.1

3. Business Value

The wide adoption of the SensorThings API and Data Model (STA) offers a solid base of existing business cases. STAplus, with its additional capabilities allows to extend existing business by realizing new use cases. The extension also offers to implement new use cases that weren't possible with the current STA v1.1.

The first part of the STAplus extension offers to operate an existing deployment with the concept of ownership: A STAplus deployment enables authenticated users the ability to manage their objects (things, sensors, observations, etc.). This upgrades a STA deployment with the ability to be run as a self-managed platform, where many different users can make their things, sensors and observations available. One example use case outside the Citizen Science context is to contribute air quality indicators or environment readings to support Smart Cities. By associating licenses on their contributions (sensor readings made available as observations), it is possible to ensure proper re-use. STAplus does not enforce a particular licensing scheme but this Best Practice document recommends to use an established, interoperable licensing on the example of Creative Commons to foster search including licensing aspects.

The second important innovation that STAplus offers is the ability to make relationships between observations explicit. The Relation concept of the extensions allows the implementation of performant convenience API functions that support semantic queries. For example, a SPARQL API could be implemented to produce linked data. The generation of relations on an existing STA deployment improves not only performance; it also allows to make relations explicit that otherwise are either difficult or impossible to be detected from another application interacting with the API.

Third and finally, the Group concept of STAplus enables to create bags or sets by creating collections of observations either via linking or deep copy. This concept offers to associate a license and an owner on a group to make it a well-defined and referenceable container of observations that can be leveraged in research or participate in other workflows.

3.1 Motivation

The definition of the SensorThings extension "plus" is based on different use cases and requirements that origins in Citizen Science. Within the H2020 project Cos4Cloud, we were facing the challenge to apply FAIR principles to an architecture that allows one application to access different APIs from Citizen Science portals for the purpose of allowing experts to verify contributed observations and leave comments. Regarding the technical implementation we had two options: Implement adapters for all different APIs and their data models or establish one common API and a general-purpose data model that provides seamless access. The decision was to develop an extension to the OGC SensorThings API and Data Model as that seemed to fit quite well.

In order to apply the SensorThings API and in particular the Data Model to be used in Citizen Science, there is a fundamental difference in its use: With Citizen Science, a SensorThings API deployment would have to support the Concept of Ownership: A user (the Citizen Scientist in our case) owns data objects and has the right to manage them. This ownership concept is important as it determines the ability of users to create, update and delete their entities (instances of the classes of the data model).

Even though the ability to support the "my" concept with STAplus, it shall not be possible that a user can impersonate another user and thereby create, update or delete data objects in an unauthorized manner. Similar, it shall not be possible that a user modifies or even deletes data objects that they don't own. The proper handling of create, update and delete is business logic specific and implementation detail that cannot and should not be captured in the data model just by itself.

Because of the rich API, standardized by the SensorThings standard, which actually defines GET, POST, PUT, PATCH and DELETE methods on an OASIS concept of OData, we came to the conclusion that "just" extending the data model and defining a business logic for Citizen Science would be sufficient.

The following example use cases illustrate the functional requirements, the data storage requirements and functionality for the STAplus extension. These use cases shall serve as test cases to validate if the STAplus data model and the SensorThings API with specific business logic can be realized.

3.2 The Camera Trap Use Case

The Camera Trap is a piece of hardware equipment with multiple sensors that automatically detects animals and captures a short video or photo of their appearance. In addition, there are sensors for measuring environmental data like temperature, lumination, humidity, etc. but also the GPS location.

From a data model perspective, a company could purchase some number of Camera Traps and make them available to Citizen Scientists (lend or rent out). The company also operates the Animal Platform upload link based on a deployed STAplus API. Then, users could operate the Camera Traps to produce observations that are stored under the provided upload link. But the users would like to make sure that their contributions are licensed, etc.

3.3 Biodiversity Use Case Natusfera/iNaturalist

Natusfera is a citizen science portal of the iNaturalist community in Spain, promoted by the Spanish National Node of GBIF (Global Biodiversity Information Facility) and CREAF. It is a free web platform and application (https://datos.gbif.es/?lang=en) created to record, organize and share observations of nature, with the aim to support the participation of nature enthusiasts and promote knowledge about the natural world. Users can participate in uploading their observations of species (images or audios), which can be identified by themselves or other users and also confirmed by others.

This use case illustrates functional requirements for importing observation data from the Natusfera portal, whose API is the GBIF based on biodiversity domain, as well as scripts for transforming data objects to the SensorThings API to be exported into STAplus data model automatically. One particular verification focuses on the use of STAplus Relation to reach the same expressiveness as for the Natusfera original API.

3.4 Biodiversity Use Case Pl@ntNet as a Service

Pl@ntNet is a worldwide citizen observatory centered on an image-based plants identification system. It is a French research and citizen science project, initially supported by Agropolis Foundation, and developed since 2009 within the framework of a consortium bringing together Cirad, INRAE, Inria, and IRD.

It is available through a mobile app and a website. Users with an account can share their plants observations under Creative Commons licence, vote on images quality, vote or suggest new determinations for existing observations, report identification errors, suggest common names in their language or give textual feedback.

This use case is about sharing Pl@ntNet observation data with partners in the context of Cos4Cloud project, such as portals that gather contributions from different citizen observatories, and in a future step, receiving feedback in the form of votes and comments. It illustrates the concept of "bag of observations" by associating one or more camera pictures to a taxonomic determination and a set of plant organs (one for every picture). It is also relevant as a query performance test, given the dataset aggregates more than 10 million "observations bags" (around 165 M tuples in PostgreSQL).

3.5 Community Process Use Case(s)

From an IoT perspective the SensorThings API provides a good and simple enough model to store observation data continuously fed by autonomous devices deployed somewhere in the wild. However, in the Cos4Cloud project we are dealing not alone with the aspect of a human observer who is providing observations, but also with a whole community who may interact with the data.

All contributors unite under a so-called Citizen Observatory (CO). Such a CO is operated as a platform for multiple users who do more than just searching and displaying data. Users may add content, search and work with such content, or even create derived work from that content. The users of a CO (members) usually interact in certain workflows, like uploading new data, verifying and linking data, quality tagging, or even assembling particular data sets.

For this to happen, the platform has to aid and stimulate a living culture within the community to enhance observations uploaded by its members. In addition, features beyond the pure collection of data are of interest for a CO.

Some examples are listed below:

- increase quality of outcome
    - linking similar/same species
    - reasoning, discussions around observations
- leverage scientific usage of data
    - permanent links for reproducibility
    - semantic linking contributions
    - assemble data collections
- foster contributions
    - acknowledgements (scientific references)
    - gamification (earn badges, explore own statistics)
    - establish networking, friendships
    - compare statistics

The STAplus provides extension points to the core STA data model where such a community process can be modeled along the observation data being collected.

4. Data Model

STAplus model is a 100% backwards-compatible extension to the SensorThings Data Model v1.1. As defined in this best practice document, the data model extension is designed to improve the applicability of the FAIR principles to existing SensorThings deployments.

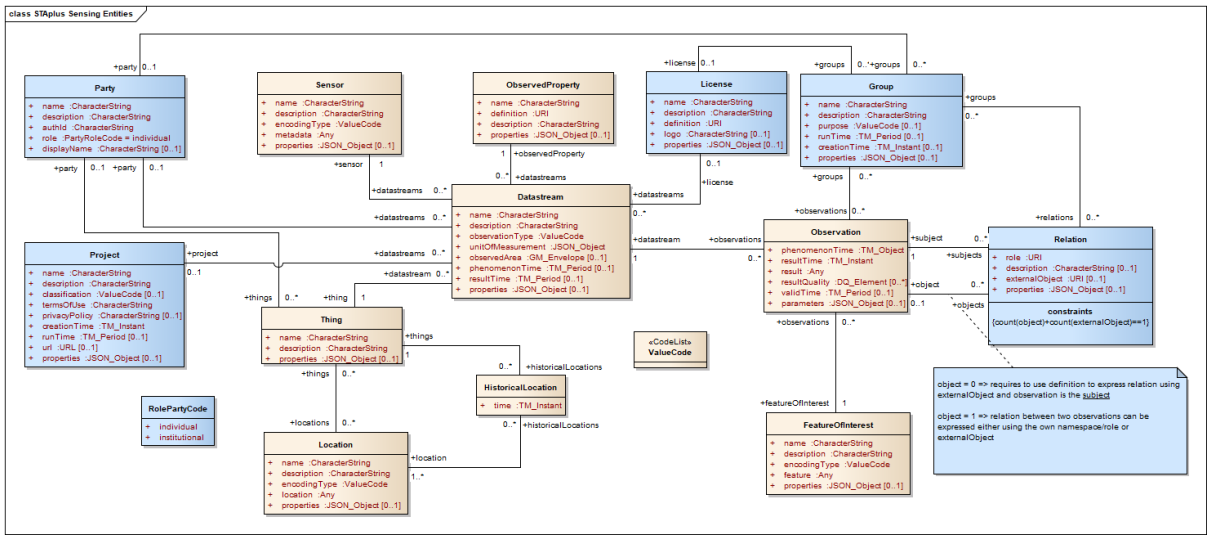

Figure 2: STAplus extension to Datastream

The figure above illustrates the STAplus data model and outlines the extension by the blue classes (see the section 1.5 for more details on the classes added). All hooks into the original SensorThings data model have the cardinality of 0.. which implies the optional use as an extension.

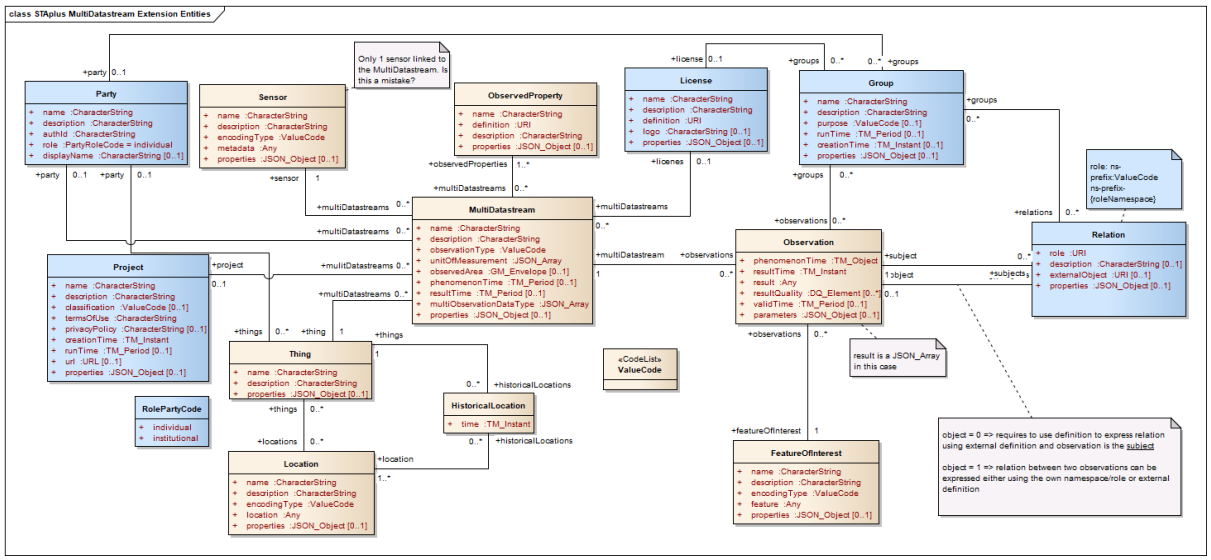

Figure 3: STAplus extension to MultiDatastream

The Figure 3 illustrates the extension to the SensorThings data model with the MultiDatastream class. Essentially, the extension (blue boxes) adds the same classes as for the Datastream case.

Even though it is possible to leverage just some classes from the extension independently, certain implications towards combined use in an implementation result from the business logic. For example, when instantiating the class Party, it is recommended to require authentication for API access. As explained in a later section, there is an overall business logic and best practice that should be considered when implementing the STAplus extension.

5. Best Practices

The SensorThings and STAplus data model guide how to structure data. The API defines a protocol for interacting with an endpoint that instantiates the data model. Beside obvious functional requirements result from implementing the data model (relations and their cardinality), different best practice can be identified when working with a deployed instance of the STAplus data model and API. The following sub-section illustrate best practice with regards to data modelling, interacting with the API and how to realize the ownership concept.

5.1 Best Practice when modelling the Camera Trap Use Case

The objective for this section is to illustrate how to model data for the Camera Trap Use Case and provide best practice when communicating with the STAplus endpoint to upload the data.

The Camera Trap App is implemented as a bash shell script. The source is available from GitHub. It segregates the interactions into a setup and a runtime phase. The setup phase instantiates necessary classes to create a working environment for operation (uploading camera trap event data).

To understand which objects the setup process must instantiate, we can first take a look at the physical sensors mounted onto the Camera Trap:

- Camera: Sony IMX 219 PQ CMOS image sensor in a fixed-focus module with IR blocking filter
- Environmental Sensors: Universal Environment Sensor Board (sensor board) measuring temperature, humidity and air pressure
- GPS Sensor: detecting GPS location
- Raspberry Pi: System date + time

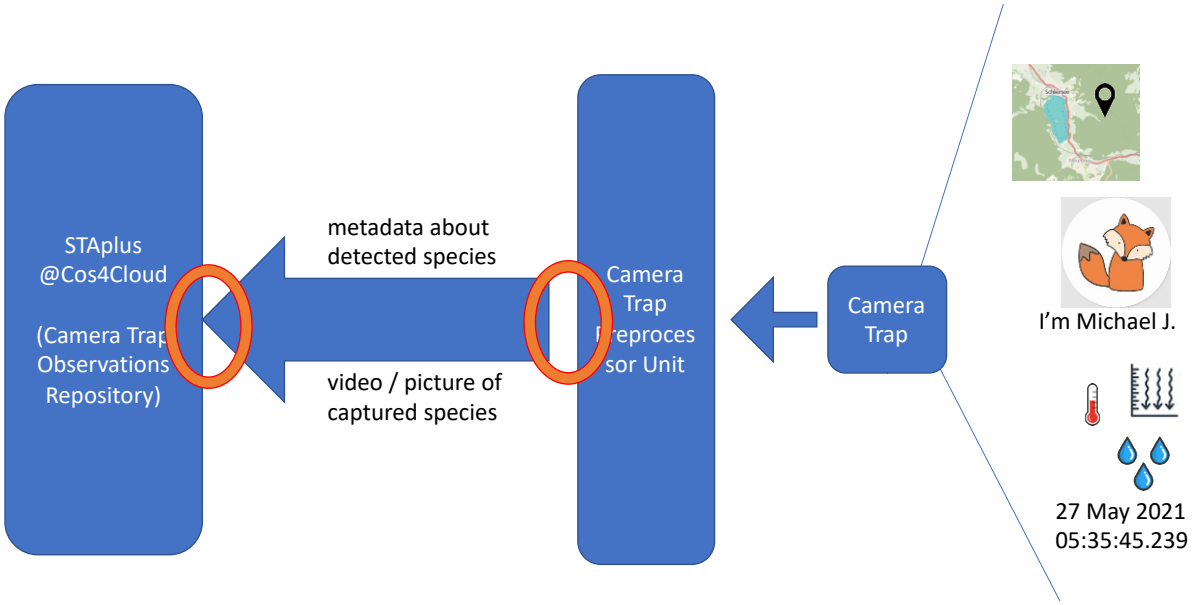

Figure 4: Data captured by the Camera Trap

Modeling the Camera Trap with STAplus results in the following (example) instantiation of the data model:

- A physical thing (the RaspberryPi) and the sensor board can be represented by an instantiation of the Thing class. Each thing is associated to the current user, operating the Camera Trap.
- Each sensor gets represented either by one single, or multiple instances.
- For each different observed property (photo, taxon, environmental data, etc.) a Datastream / MultiDatastream instance gets created by the user. The Party instance, representing the user, gets linked to the datastream. In case the user likes to specify licensing conditions, a license is linked to the datastream.
- For each Camera Trap event, all related observations are stored in a Group instance.
- The relations between the individual part of the observation get illustrated with instantiations of the Relation class.

The FeatureOfInterest class represents the location or area observed by the Camera Trap. In addition to the actual camera device which creates imagery data, the sensor board measures additional phenomena where the Camera Trap has been deployed. Depending on the granularity needed, imagery and phenomena data can be linked to different FeatureOfInterest instances, as the observed areas of the camera sensor and the location if the sensor board might not be the same. However, this should not be a problem, as the FeatureOfInterest class is linked to each part of the observation event itself.

5.1.1 Setup of the Camera Trap

The following walk-through is using a STAplus endpoint on localhost and leveraging the native SensorThings API protocol to illustrate the different steps.

Note: All URLs are examples on localhost which may be different upon replay. This walk-through can be optimized using the Batch-Processing API function as illustrated afterwards.

The first interaction between the App and the STAplus endpoint is to create a project that contains the overall description for operating a Camera Trap.

```
{
    "name": "Animal Detection by DynAikon Camera Trap",
    "description": "The automatic detection of species by all participating camera traps",
    "url": "https://cos4cloud.secd.eu/projects/cameratrap",
    "termsOfUse": "Please do not upload sensitive information!.",
    "privacyPolicy": "This project stores the user's globally unique identifier that cannot be
used to retrieve personal information.",
    "creationTime": "2021-05-28T08:12:00Z",
    "classification": "public"
}
```

Example 1: Request for creating a project (HTTP POST to /Projects)

The CT-LoaderApp stores the Project@iot.id returned by the STAplus endpoint on local disk drive. This ensures proper re-use for each execution of the Camera Trap (runtime process).

The next interaction with the STAplus endpoint creates the thing (Raspberry Pi and Environmental Sensor Board) with its datastream.

```
{
    "name": "RaspberryPi",
    "description": "Raspberry Pi 4 Model B, 4x 1,5 GHz, 4 GB RAM, WLAN, BT is the latest
product in the popular Raspberry Pi range of computers",
    "properties": {
        "CPU": "1.4GHz",
```

```
        "RAM": "4GB"
    },
    "Party": {
        "authId": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea",
        "displayName": "Long John Silver",
        "description": "The opportunistic pirate by Robert Louis Stevenson",
        "role": "individual"
    },
    "Datastreams": [
        {
            "unitOfMeasurement": {
                "name": "n/a",
                "symbol": "",
                "definition": "https://www.merriam-webster.com/dictionary/picture"
            },
            "name": "photo datastream",
            "description": "this datastream is about pictures",
            "observationType": "http://www.opengis.net/def/observationType/OGC-
OM/2.0/OM_Measurement",
            "ObservedProperty": {
                "name": "Picture",
                "definition": "https://www.merriam-webster.com/dictionary/picture",
                "description": "The image taken by the camera (the sensor)"
            },
            "Sensor": {
                "name": "Pi NoIR - Raspberry Pi Infrared Camera Module",
                "description": "Sony IMX 219 PQ CMOS image sensor in a fixed-focus module with
IR blocking filter removed",
                "encodingType": "application/pdf",
                "metadata": "https://cdn-reichelt.de/documents/datenblatt/A300/RASP_CAN_2.pdf"
            },
            "License": {"@iot.id": "1"},
            "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
            "Project": {"@iot.id": "1"}
        },
        {
            "unitOfMeasurement": {
                "name": "GBIF Identity",
                "symbol": "n/a",
                "definition": "https://www.gbif.org/species"
            },
            "name": "GBIF Identifier for Species",
            "description": "The GBIF identifiers for species",
            "observationType": "GBIF Taxonomy",
            "ObservedProperty": {
                "name": "Taxon",
                "definition": "https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-
bb099caae36c",
                "description": "GBIF Backbone Taxonomy"
            },
            "Sensor": {
                "name": "DynAikon AI for automatic species detection",
                "description": "The DynAikon automatic species detection",
                "encodingType": "text/html",
                "metadata": "https://DynAikon.com/"
            },
            "License": {"@iot.id": "2"},
            "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
            "Project": {"@iot.id": "1"}
        }
    ]
}
```


Example 2: Request to create the Raspberry Pi thing and its datastream (HTTP POST /Things)

The request to create the Raspberry Pi thing contains the relevant datastream inline and the links to License and Party instances. Also, the datastream links to the project created before. The associated party is represented by the user's unique identifier as it will be resolved by the STAplus endpoint.

```
{
    "name": "Universal Environment Board",
    "description": "This board measures air temperature, humidity and pressure",
    "properties": {
        "Temperature": "temperature on board",
        "Humidity": "air humidity",
        "Pressure": "air pressure sensor",
        "GPS": "GPS unit available"
    },
    "MultiDatastreams": [
        {
            "name": "Environmental Datastream from Camera Trap",
            "description": "Environment data for air temperature, humidity, pressure",
            "multiObservationDataTypes": [
                "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
                "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
                "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement"
            ],
            "observationType": "http://www.opengis.net/def/observationType/OGC-
OM/2.0/OM_ComplexObservation",
            "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
            "properties": {
                "fieldOne": "Temperature",
                "fieldTwo": "Humidity",
                "fieldThree": "Preasure",
                "uuid": "7c9e768c4-1201-4fcb-81d4-8be29e16f522"
            },
            "unitOfMeasurements": [
                {
                    "name": "Temperature",
                    "symbol": "C",
                    "definition":
"http://www.qudt.org/qudt/owl/1.0.0/qudt/index.html#TemperatureUnit"
                },
                {
                    "name": "Humidity",
                    "symbol": "RH",
                    "definition": "https://byjus.com/physics/unit-of-humidity/"
                },
                {
                    "name": "Pressure",
                    "symbol": "mbar",
                    "definition": "https://en.wikipedia.org/wiki/Atmospheric_pressure"
                }
            ],
            "Sensor": {
                "name": "Environment Sensor",
                "description": "This sensor produces temperature, humidity and pressure",
                "encodingType": "text/html",
                "metadata": "https://google.de",
                "properties": {"calibrated": "2021-1-16T12:00:00Z"}
            },
            "ObservedProperties": [
                {
                    "name": "DegC",
                    "definition": "https://en.wikipedia.org/wiki/Temperature",
                    "description": "Air Temperature in Celcius"
                },
                {
                    "name": "Relative Air Humidity",
                    "definition": "https://en.wikipedia.org/wiki/Humidity",
                    "description": "Air Humidity"
                },
                {
                    "description": "Atmospheric pressure",
                    "definition": "https://en.wikipedia.org/wiki/Atmospheric_pressure",
                    "name": "Atmospheric pressure"
                }
            ],
            "Project": {"@iot.id": 1}
        }
    ]
}
```


Example 3: Request to create the sensor board thing and its MultiDatastream instance (HTTP POST /Things)

The response from the STAplus endpoint contains the URL location for the created thing. E.g. *http://localhost:8080/FROST-Server.HTTP/v1.1/Things(17)*

From that URL, it is possible to extract the ids for each created datastream. One implementation specific approach used here stores the ids as property 'uuid'. With that, the datastream can be obtained via the following example URL.

```
http://localhost:8080/FROST-Server.HTTP/v1.1/Things(17)/Datastreams?
    $filter=properties/uuid%20eq%20%27c9e768c4-1201-4fcb-81d4-8be29e16f522%27
```

Example 4: Fetch the datastream identified by UUID

To keep the footprint of the response small, it would be sufficient to request the iot.id only.

```
http://localhost:8080/FROST-Server.HTTP/v1.1/Things(17)/Datastreams?
    $select=@iot.id&filter=properties/uuid%20eq%20%27c9e768c4-1201-4fcb-81d4-8be29e16f522%27
```

Example 5: Fetch the datastream iot.id by UUID

The approach above requires one individual request to the STAplus endpoint for each Datastream object that got generated. However, it is possible to obtain the identifiers with one single request from which the iot.id values can be extracted. This request URL is extended by /Datastreams and uses the $select option provided by the API:

```
http://localhost:8080/FROST-Server.HTTP/v1.1/Things(17)/Datastreams?
    $select=@iot.id,name,properties/uuid
```

Example 6: Fetch all datastream iot.id and uuid property for the thing


```
{
    "value": [
        {
            "@iot.id": 16,
            "name": "photo datastream",
            "properties": {"uuid": "45bad1ff-7146-41af-bde0-23b6f795943b"}
        },
        {
            "@iot.id": 17,
            "name": "GBIF Identifier for Species",
            "properties": {"uuid": "c9e768c4-1201-4fcb-81d4-8be29e16f522"}
        }
    ]
}
```


Example 7: Response of iot.id and uuid properties for datastream associated to Thing(17)

Saving the datastream ids and their application specific UUIDs to local storage allows a performant posting of observations during runtime.

The same procedure is followed to instantiate the thing for the sensor board. The only difference is that the environmental data gets associated to a MultiDatastream instance.

The URL to fetch the @iot.id for the created multi-datastream is similar to the URL before:

http://localhost:8080/FROST-Server.HTTP/v1.1/Things(18)/MultiDatastreams?$select=@iot.id,name,properties/uuid

```
{
    "value": [
        {
            "@iot.id": 1,
            "name": "Environmental Datastream from Camera Trap",
            "properties": {"uuid": "ef500bdd-5117-49d0-94d4-d9a5f43e23a0"}
        }
    ]
}
```

Example 8: Response of the multi-datastream for the sensor board

All @iot.id values are stored to disk. This allows repeated start and stop of the Camera Trap to use the same thing, sensor, datastream and multi-datastream.

5.1.2 Camera Trap at Runtime

The Figure 5 below illustrates one approach how to map the different data objects created by the Camera Trap to the STAplus data model.

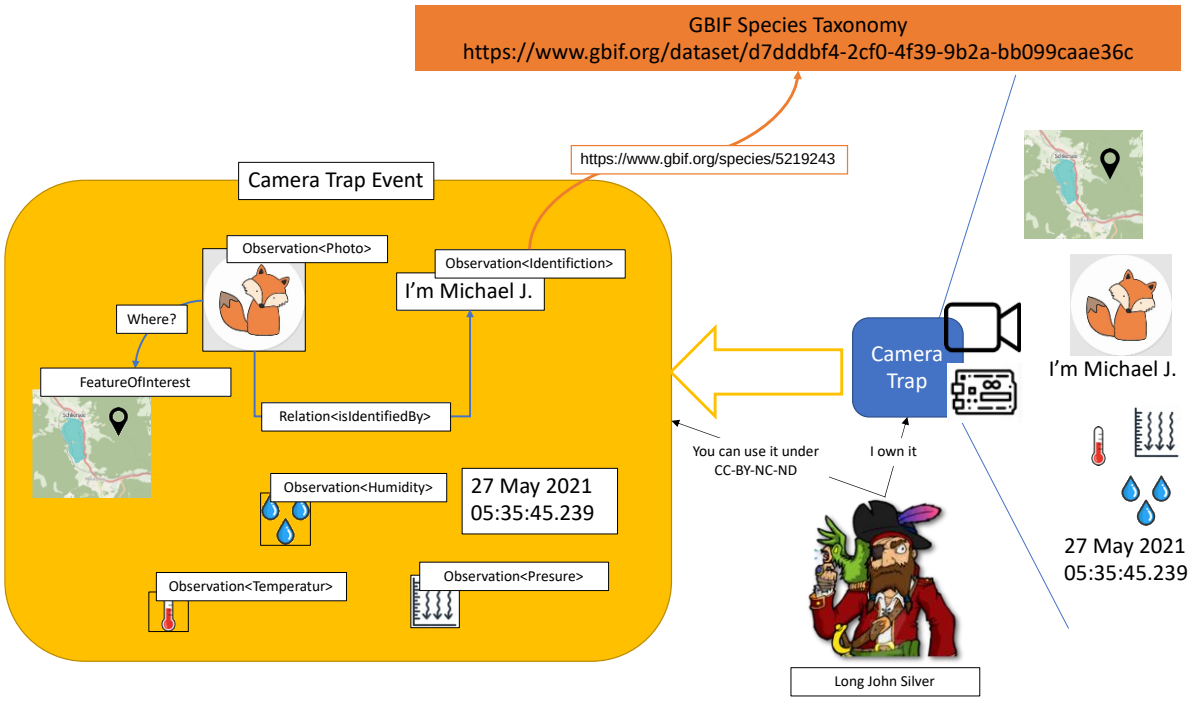

Figure 5: Mapping the trap event data to STAplus (overview)

For the runtime, the CT-LoaderApp must determine (perhaps by interacting with the user) how to represent the feature-of-interest. In the simplest case, the FeatureOfInterest instance captures the location only. However, for the camera trap, it is good practice to create two different features of interest: one for the animal and another for the air.
Assuming that the location of the detected animal (slightly) differs for each trap event and is not a point but an area, the runtime process would create a new feature-of-interest for each observation exposed from the taxon and photo datastream.

```
{
    "name": "animal",
    "description": "The location vicinity of the animal being detected",
    "encodingType": "application/geo+json",
    "feature": {
        "type": "Polygon",
        "coordinates": [
            [
                [
                    11.510560,
                    48.112145
                ],
                [
                    11.512809,
                    48.112483
                ],
                [
                    11.513455,
                    48.109776
                ],
                [
                    11.511761,
                    48.108787
                ],
                [
                    11.510609,
                    48.110516
                ],
                [
                    11.510560,
                    48.112145
                ]
            ]
        ]
    }
}
```


Example 9: FeatureOfInterest instance for the animal detected at a Camera Trap event

The feature-of-interest for the air is at the location of the camera trap. Therefore, it would be sufficient to produce one FeatureOfInterest instance when the Camera Trap is activated. The created feature-of-interest could then be linked to the multi-datastream that holds the air measurements.


```
{
  "name": "air",
  "description": "The location of the air measurements",
  "encodingType": "application/geo+json",
  "feature": {
    "type": "Point",
    "coordinates": [
      2.044367,
      41.485526
    ]
  }
}
```


Example 10: FeatureOfInterest instance of the sensor measurements for a Camera Trap event

Uploading observations for a camera trap event requires a sequence of interactions:

1) The photo or video of the detected animal must be uploaded first using the convenience API ObservationUpload (/$upload). The returned URL contains the id for the created observation.
2) The group of observations gets uploaded. The group contains the taxon and air observations, the feature-of-interest for the animal and the link to the feature-of-interest for the air measurements.

3) The relations on the previously generated observations get uploaded. As this request requires the observation ids from the previous request, a read request to the created group must be done.

Uploading the binary observation representing the animal's photo or short video can be achieved using the UploadObservation API extension.

```
{
    "phenomenonTime": "2021-05-28T02:45:00Z",
    "resultTime": "2021-05-28T08:45:00Z",
    "result": "",
    "parameters": {
        "tilt_angle": "30",
        "distance": "5",
        "shutter": "2.4",
        "speed": "1/400"
    },
    "FeatureOfInterest": {"@iot.id": "1"},
    "Datastream": {"@iot.id": "16"}
}
```

Example 11: Observation representing the animal (part one)

The processing of the /$observation endpoint[1] first stores the binary, then generates the HTTP access URL and replaces the value of the result property with the URL.

```
curl --verbose --header "Authorization: Bearer b5635cd2a8e2766636a08c761bccbb3282b3aa97" --
form "observation=<photo.json;type=application/json" --form "file=@tongue.png"
"http://localhost:8080/FROST-Server.HTTP/v1.1/\$observation"
```

Example 12: Example POST request using CURL to upload binary observation

The request is of type HTTP multipart where the first part is the JSON representing the SensorThings observation (example 11) and the second part is the image (tongue.png).

```
{
    "name": "Gray Fox",
    "description": "Gray Fox Camera Trap Event",
    "created": "2021-04-22T18:10:00Z",
    "runtime": "2021-04-21T12:00:00Z/2021-04-22T15:43:00Z",
    "License": {"@iot.id": "3"},
    "Observations": [
        {"@iot.id": "1"},
        {
            "phenomenonTime": "2021-04-21T12:00:00Z",
            "resultTime": "2021-04-22T15:43:00Z",
            "result": "https://www.gbif.org/species/5219243",
            "FeatureOfInterest": {"@iot.id": "1"},
            "Datastream": {"@iot.id": "15"},
            "parameters": {
                "uuid": "73d4de15-bace-4a46-8dcb-509a1970a475"
            }
        },
        {
            "phenomenonTime": "2020-05-26T23:00:00.000Z/2020-05-27T23:00:00.000Z",
            "resultTime": "2021-04-22T15:43:00Z",
            "FeatureOfInterest": {"@iot.id": "3"},
            "result": [
                1.3,
                87.5,
                980.02
```

[1] The $observation endpoint uses an extension of the SensorThings API that allows to upload a binary observation.

```
            ],
            "MultiDatastream": {"@iot.id": "1"},
            "parameters": {
                "uuid": "56749785-2331-4242-916e-e7086054d1cd"
            }
        }
    ]
}
```

Example 13: Generating the group (each observation has its own uuid property)

Next step is to fetch the STAplus generated iot.ids for the observation photo and taxon, because these are being linked in the next step.

```
http://localhost:8080/FROST-Server.HTTP/v1.1/Groups(6)/Observations?
    $select=@iot.id,parameters/uuid
```

Example 14: Fetch all observation iot.id and uuid properties for the created group

```
{
    "value": [
        {
            "@iot.id": 1,
            "parameters": {}
        },
        {
            "@iot.id": 18,
            "parameters": {"uuid": "73d4de15-bace-4a46-8dcb-509a1970a475"}
        },
        {
            "@iot.id": 19,
            "parameters": {"uuid": "56749785-2331-4242-916e-e7086054d1cd"}
        }
    ]
}
```

Example 15: Response of iot.id and uuid properties for observations associated to Group(1)

Based on the server generated @iot.id identifiers, any relation can be generated for supporting semantic queries. Relations for this example express:

- The photo is about the animal “red fox”, identified by taxon https://www.gbif.org/species/5219243
- The user is the owner of the photo
- The Darwin core relationships of the detected species

Below is an example how to create a relation via
http://localhost:8080/FROST-Server.HTTP/v1.1/Groups(1)/Relations

```
{
    "Subject": {"@iot.id": 1},
    "Object": {"@iot.id": 18},
    "name": "taxonConceptID",
    "description": "http://rs.tdwg.org/dwc/terms/taxonConceptID",
    "role": "taxonConceptID",
    "namespace": "http://rs.tdwg.org/dwc/terms/"
}
```

Example 16: Creating a Darwin Core relation that relates the observed photo of the animal to the DWC identifier

Note: section 5.3.2 outlines how to leverage the Relation to express “Darwin Core”.

5.2 Best Practice leveraging the STA API Batch-Processing

The previous section illustrated how to realize the Camera Trap use case leveraging the standard API protocol. This section introduces the improvements based on the convenience API Batch-Processing as described (and standardized in the OGC Sensor Things API v.1.1, section 11 (https://docs.opengeospatial.org/is/15-078r6/15-078r6.html#70). In essence, the use of Batch-processing reduces the back-and-forth communication with the SensorThings endpoint tremendously.

Using the Batch-processing, the Camera Trap use case can essentially be split into two parts:

1) Initialization: The CT-LoaderApp creates the relevant entities to allow processing of generated camera trap events data.
2) Runtime: The CT-LoaderApp uploads all data that belongs to a camera trap event.

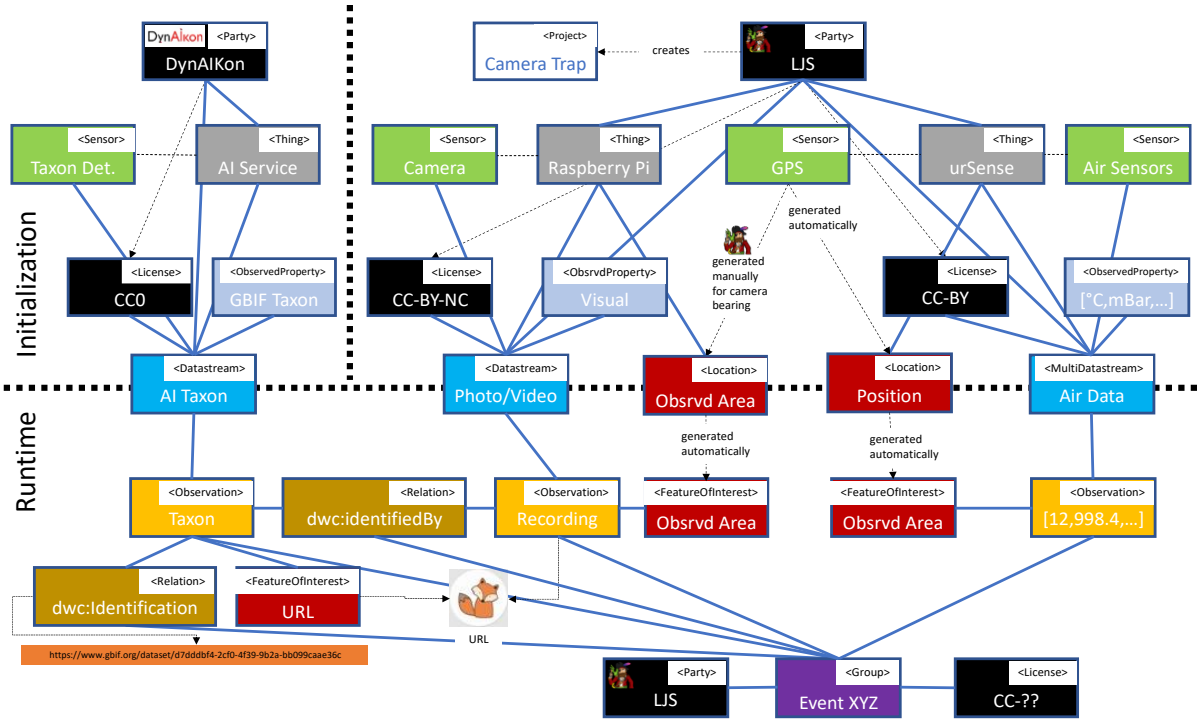

Figure 6: Entities, their linking and separation into initialization and runtime

Figure 6 illustrates that there are two actors: DynAikon and Long John Silver. DynAikon, as an institutional party, creates a datastream that represents the results for species determination via their "AI Service". That AI Service is used by the CT-LoaderApp to automatically determine (guess) the animal captured on the recording.
Long John Silver (LJS) is the operator of the Camera Trap. With running the setup, he must specify the area of interest for the camera. This area can be created, for example, by simply drawing a polygon on a map, and copying and pasting the GeoJSON geometry into a config file. Also, LJS must specify the license for the visual capture (photo or video) and the air data that gets produced by the sensor board.

5.2.1 Camera Trap Initialization using Batch-Processing

As outlined above, the initialization phase requires to instantiate different classes that provide the backbone for uploading camera trap event data during runtime. Some instances get

created when a user first starts the camera trap. Other instances get created / updated when the user moves the camera trap from one location to another.

First initial start

- Party: Representing the acting user.
- Project: The overall container that describes the activity, e.g. “Animal detection in my garden”.
- Thing (Raspberry Pi): The computer that runs the Camera Trap and the STAplus uploader application; Linked to Party
- Location: Area observed by the camera
- Sensor Camera: Produces the photo or video of detected animal; Linked to Thing (Raspberry Pi)
- Datastream (Photo / Video): Linked to the Raspberry Pi, Photo Datastream and License; will be referenced during runtime by uploaded photo/video of detected animal
- Thing (EnvBoard): The HW board that measures the environment data and determines the current GPS location; Linked to Party
- Location: Linked to the EnvBoard representing the current position
- Sensor (EnvBoard): Produces the air measurements and GPS location for the camera trap location; Linked to Thing (EnvBoard); Linked to Party
- MultiDatastream (Environment data): Linked to Environment Board and License; will be referenced during runtime by uploaded environment data


```
{
    "requests": [
        {
            "id": "myProject",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Projects",
            "body": {
                "name": "Animal Detection by DynAikon Camera Trap",
                "description": "The automatic detection of species by all participating camera
traps",
                "url": "https://cos4cloud.secd.eu/projects/cameratrap",
                "termsOfUse": "Please do not upload sensitive information!.",
                "privacyPolicy": "This project stores the user's globally unique identifier
that cannot be used to retrieve personal information.",
                "created": "2021-05-28T08:12:00Z",
                "classification": "public"
            }
        },
        {
            "id": "thingRasPi",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Things",
            "body": {
                "name": "RaspberryPi",
                "description": "Raspberry Pi 4 Model B, 4x 1,5 GHz, 4 GB RAM, WLAN, BT is the
latest product in the popular Raspberry Pi range of computers",
                "properties": {
                    "CPU": "1.4GHz",
                    "RAM": "4GB"
                },
                "Party": {
                    "authId": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea",
                    "nickName": "Long John Silver",
                    "description": "The opportunistic pirate by Robert Louis Stevenson",
                    "role": "individual",
                    "properties": {"sub": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"}
                }
```

```
            }
        },
        {
            "id": "dsPicture",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Datastreams",
            "body": {
                "unitOfMeasurement": {
                    "name": "n/a",
                    "symbol": "",
                    "definition": "https://www.merriam-webster.com/dictionary/picture"
                },
                "name": "photo datastream",
                "description": "this datastream is about pictures",
                "observationType": "http://www.opengis.net/def/observationType/OGC-
OM/2.0/OM_Measurement",
                "properties": {"uuid": "45bad1ff-7146-41af-bde0-23b6f795943b"},
                "ObservedProperty": {
                    "name": "Picture",
                    "definition": "https://www.merriam-webster.com/dictionary/picture",
                    "description": "The image taken by the camera (the sensor)"
                },
                "Sensor": {
                    "name": "Pi NoIR - Raspberry Pi Infrared Camera Module",
                    "description": "Sony IMX 219 PQ CMOS image sensor in a fixed-focus module
with IR blocking filter removed",
                    "encodingType": "application/pdf",
                    "metadata": "https://cdn-
reichelt.de/documents/datenblatt/A300/RASP_CAN_2.pdf"
                },
                "Thing": {"@iot.id": "$thingRasPi"},
                "License": {"@iot.id": 1},
                "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
                "Project": {"@iot.id": "$myProject"}
            }
        },
        {
            "id": "dsIdentity",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Datastreams",
            "body": {
                "unitOfMeasurement": {
                    "name": "GBIF Identity",
                    "symbol": "n/a",
                    "definition": "https://www.gbif.org/species"
                },
                "name": "GBIF Identifier for Species",
                "description": "The GBIF identifiers for species",
                "observationType": "GBIF Taxonomy",
                "properties": {"uuid": "c9e768c4-1201-4fcb-81d4-8be29e16f522"},
                "ObservedProperty": {
                    "name": "Taxon",
                    "definition": "https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-
bb099caae36c",
                    "description": "GBIF Backbone Taxonomy"
                },
                "Sensor": {
                    "name": "DynAikon AI for automatic species detection",
                    "description": "The DynAikon automatic species detection",
                    "encodingType": "text/html",
                    "metadata": "https://DynAikon.com/"
                },
                "Thing": {"@iot.id": "$thingRasPi"},
                "License": {"@iot.id": 2},
                "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
                "Project": {"@iot.id": "$myProject"}
            }
        },
        {
            "id": "thingEnvBoard",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Things",
            "body": {
```

```
                "name": "Universal Environment Board",
                "description": "This board measures air temperature, humidity and pressure",
                "properties": {
                    "Temperature": "temperature on board",
                    "Humidity": "air humidity",
                    "Pressure": "air pressure sensor",
                    "GPS": "GPS unit available"
                }
            }
        },
        {
            "id": "mdsEnvironment",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "MultiDatastreams",
            "body": {
                "name": "Environmental Datastream from Camera Trap",
                "description": "Environment data for air temperature, humidity, pressure",
                "multiObservationDataTypes": [
                    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
                    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
                    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement"
                ],
                "observationType": "http://www.opengis.net/def/observationType/OGC-
OM/2.0/OM_ComplexObservation",
                "properties": {
                    "fieldOne": "Temperature",
                    "fieldTwo": "Humidity",
                    "fieldThree": "Presure",
                    "uuid": "7c9e768c4-1201-4fcb-81d4-8be29e16f522"
                },
                "unitOfMeasurements": [
                    {
                        "name": "Degree Celcius",
                        "symbol": "C",
                        "definition":
"http://www.qudt.org/qudt/owl/1.0.0/qudt/index.html#TemperatureUnit"
                    },
                    {
                        "name": "Percent",
                        "symbol": "%",
                        "definition": "https://byjus.com/physics/unit-of-humidity/"
                    },
                    {
                        "name": "Millibar",
                        "symbol": "mbar",
                        "definition": "https://en.wikipedia.org/wiki/Atmospheric pressure"
                    }
                ],
                "Sensor": {
                    "name": "Environment Sensor",
                    "description": "This sensor produces temperature, humidity and pressure",
                    "encodingType": "text/html",
                    "metadata": "https://google.de",
                    "properties": {"calibrated": "2021-1-16T12:00:00Z"}
                },
                "ObservedProperties": [
                    {
                        "name": "DegC",
                        "definition": "https://en.wikipedia.org/wiki/Temperature",
                        "description": "Air Temperature in Celcius"
                    },
                    {
                        "name": "Relative Air Humidity",
                        "definition": "https://en.wikipedia.org/wiki/Humidity",
                        "description": "Air Humidity"
                    },
                    {
                        "description": "Atmospheric pressure",
                        "definition": "https://en.wikipedia.org/wiki/Atmospheric_pressure",
                        "name": "Atmospheric pressure"
                    }
                ],
                "Thing": {"@iot.id": "$thingEnvBoard"},
                "License": {"@iot.id": 2},
                "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
```

```
                "Project": {"@iot.id": "$myProject"}
            }
        }
    ]
}
```

Example 17: Batch-Processing initialization example for the STAplus endpoint to setup the Camera Trap (*https://gist.github.com/hylkevds/83cde8c4b8b561ffbab12bc1bb594251*)

Moving the camera trap from one location to another

- Location (Observed Area): Updated based on the new location and bearing of the camera. Update Thing(Raspberry Pi)->Location
- Location (EnvBoard): The software must automatically update the location. Update Thing(EnvBoard)->Location

```
{
    "requests": [
        {
            "id": "LocationCamera",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Locations",
            "body": {
                {
                    "name": "animal",
                    "description": "The location vicinity of the animal being detected",
                    "encodingType": "application/geo+json",
                    "location": {
                        "type": "Polygon",
                        "coordinates": [
                            [
                                [ 11.610560, 48.212145 ],
                                [ 11.612809, 48.212483 ],
                                [ 11.613455, 48.209776 ],
                                [ 11.611761, 48.208787 ],
                                [ 11.610609, 48.210516 ],
                                [ 11.610560, 48.212145 ]
                            ]
                        ]
                    },
                    "Things": {"@iot.id": 1}
                }
            }
        },
        {
            "id": "LocationEnvSensor",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Locations",
            "body": {
                {
                    "name": "air",
                    "description": "The location of the air measurements",
                    "encodingType": "application/geo+json",
                    "feature": {
                        "type": "Point",
                        "coordinates": [ 2.044367, 41.485526 ]
                    },
                    "Things": {"@iot.id": 2}
                }
            }
        }
    ]
}
```

Example 18: Batch-Processing update example for the STAplus endpoint once the Camera Trap setup has changed

The above example illustrates updating location and camera bearing, assuming the camera and sensor board have things with ids 1 and 2.

5.2.2 Camera Trap Runtime using Batch Processing

During runtime, the CT-LoaderApp uploads all data that belongs to an event for animal detection. The data consists of the GPS location of the sensor board, the actual environment data, the photo/video of the detected animal.

- FeatureOfInterest (EnvBoard): The (point) location for the measured environment data; automatically generated from Thing(EnvBoard)->Location
- Observation (EnvBoard): The environment data; Uploaded as observation and linked to the multi-datastream
- FeatureOfInterest (Recording): The (polygon) area that was observed by the camera and in which the animal was detected; Can automatically be generated from the Thing(Raspberry Pi)->Location
- Observation (photo/video): Linked to visual datastream
- Observation (species guess): Linked to identification datastream
- FeatureOfInterest(species guess): Needs to be generated for each visual observation linking the species prediction observation to the actual URL for the photo/video
- Relation I: link photo/video with identification
- Relation II: link identification with Darwin Core identifier
- Group: Represents all the data that belongs to the event of a species detection; Container for all the entities above


```
{
    "requests": [
        {
            "id": "FoIAnimal",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "FeaturesOfInterest",
            "body": {
                "name": "animal",
                "description": "The location vicinity of the animal being detected",
                "encodingType": "application/geo+json",
                "feature": {
                    "type": "Polygon",
                    "coordinates": [
                        [
                            [
                                11.510560,
                                48.112145
                            ],
                            [
                                11.512809,
                                48.112483
                            ],
                            [
                                11.513455,
                                48.109776
                            ],
                            [
                                11.511761,
                                48.108787
                            ],
                            [
                                11.510609,
                                48.110516
                            ],
                            [
                                11.510560,
                                48.112145
                            ]
                        ]
                    ]
                }
```

```
            }
        },
        {
            "id": "obsPhoto",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Observations",
            "body": {
                "phenomenonTime": "2021-05-28T02:45:00Z",
                "resultTime": "2021-05-28T08:45:00Z",
                "result": "http://example.com/some/path.jpg",
                "parameters": {
                    "tilt_angle": "30",
                    "distance": "5",
                    "shutter": "2.4",
                    "speed": "1/400"
                },
                "FeatureOfInterest": {"@iot.id": "$FoIAnimal"},
                "Datastream": {"@iot.id": 16}
            }
        },
        {
            "id": "obsIdent",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Observations",
            "body": {
                "phenomenonTime": "2021-04-21T12:00:00Z",
                "resultTime": "2021-04-22T15:43:00Z",
                "result": "https://www.gbif.org/species/5219243",
                "FeatureOfInterest": {"@iot.id": "$FoIAnimal"},
                "Datastream": {"@iot.id": 15},
                "parameters": {"uuid": "73d4de15-bace-4a46-8dcb-509a1970a475"}
            }
        },
        {
            "id": "obsEnviro",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Observations",
            "body": {
                "phenomenonTime": "2020-05-26T23:00:00.000Z/2020-05-27T23:00:00.000Z",
                "resultTime": "2021-04-22T15:43:00Z",
                "FeatureOfInterest": {"@iot.id": 3},
                "result": [
                    1.3,
                    87.5,
                    980.02
                ],
                "MultiDatastream": {"@iot.id": 1},
                "parameters": {"uuid": "56749785-2331-4242-916e-e7086054d1cd"}
            }
        },
        {
            "id": "myGroup",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Groups",
            "body": {
                "name": "Gray Fox",
                "description": "Gray Fox Camera Trap Event",
                "created": "2021-04-22T18:10:00Z",
                "runtime": "2021-04-21T12:00:00Z/2021-04-22T15:43:00Z",
                "License": {"@iot.id": 3},
                "Observations": [
                    {"@iot.id": 1},
                    {"@iot.id": "$obsIdent"},
                    {"@iot.id": "$obsEnviro"}
                ]
            }
        },
        {
            "id": "myRelation1",
            "atomicityGroup": "group1",
            "method": "post",
            "url": "Relations",
```

```
            "body": {
                "Groups": [
                    {"@iot.id": "$myGroup"}
                ],
                "Subject": {"@iot.id": 1},
                "Object": {"@iot.id": "$obsIdent"},
                "name": "taxonConceptID",
                "description": "http://rs.tdwg.org/dwc/terms/taxonConceptID",
                "role": "taxonConceptID",
                "namespace": "http://rs.tdwg.org/dwc/terms/"
            }
        }
    ]
}
```

Example 19: Batch-Processing runtime example for uploading Camera Trap event data to the STAplus endpoint (*https://gist.github.com/hylkevds/9b88122bedc05abfc0226427ff0d26dd*)

5.3 Best Practice when applying STAplus to Natusfera/iNaturalist data

This section introduces a best practice approach for modelling data from the Natusfera portal to the STAplus data model. The documented approach introduces one mapping to STAplus but also illustrates how to interact with the API.

The objective for this best practice is to export observation data from the Natusfera portal to the STAplus endpoint by identifying the data schema used in common and compatible. Natusfera is a citizen science observation portal focused on monitoring biodiversity, whose API is based on the Global Biodiversity Information Facility (GBIF) using common terms on biodiversity domain. Due to the nature of the data model structure specific to biodiversity domain, transforming some classes and the associated attributes from Natusfera to STAplus is not straightforward, or some concepts do not simply exist in one or another domain.

As a use case, we tested transforming observation data available in Natusfera portal, using observation group ID 313411 (see Figure 7), also encoded in JSON format (see Example 20).

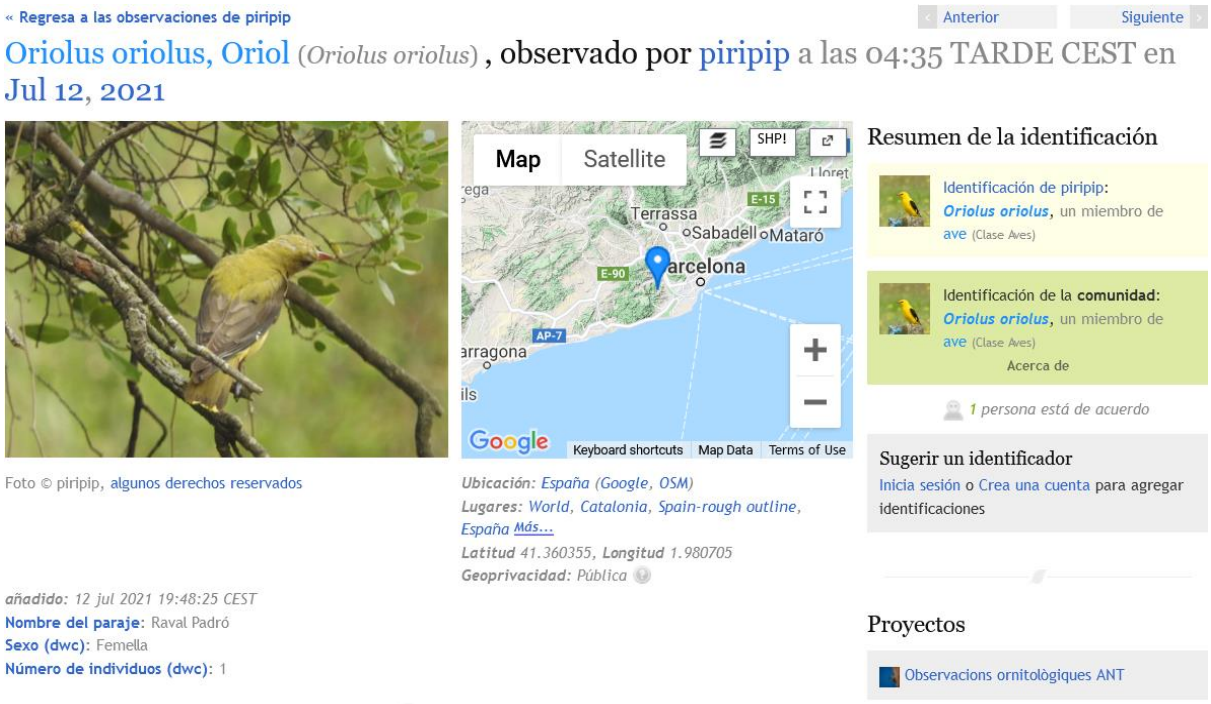

Figure 7: A screenshot of the web interface in Natusfera portal showing observation group ID 313411 (source: https://natusfera.gbif.es/observations/313411)


```
{
  "captive": false,
  "comments_count": 0,
  "community_taxon_id": 10091,
  "created_at": "2021-07-12T19:48:25+02:00",
  "delta": false,
  "description": "",
  "geoprivacy": null,
  "iconic_taxon_id": 5,
  "id": 313411,
  "id_please": false,
  "identifications_count": 2,
  "latitude": "41.360355",
  "license": "CC-BY-NC",
  "location_is_exact": false,
  "longitude": "1.980705",
  "map_scale": null,
  "mappable": true,
  "num_identification_agreements": 1,
  "num_identification_disagreements": 0,
  "oauth_application_id": null,
  "observation_photos_count": 1,
  "observation_sounds_count": 0,
  "observed_on": "2021-07-12",
  "observed_on_string": "2021-07-12 16:35:46",
  "old_uuid": null,
  "out_of_range": null,
  "place_guess": "España",
  "positional_accuracy": null,
  "positioning_device": null,
  "positioning_method": null,
  "public_positional_accuracy": null,
  "quality_grade": "research",
  "site_id": null,
  "species_guess": "Oriolus oriolus, Oriol",
  "taxon_id": 10091,
  "time_observed_at": "2021-07-12T16:35:46+02:00",
  "time_zone": "Paris",
  "timeframe": null,
  "updated_at": "2021-07-13T13:45:50+02:00",
  "uri": "http://natusfera.gbif.es/observations/313411",
```

```
  "user_id": 2319,
  "uuid": "78fdcf48-5cb1-496d-a80a-d96a12f990a2",
  "zic_time_zone": "Europe/Paris",
  "user_login": "piripip",
  "iconic_taxon_name": "Aves",
  "created_at_utc": "2021-07-12T17:48:25Z",
  "updated_at_utc": "2021-07-13T11:45:50Z",
  "time_observed_at_utc": "2021-07-12T14:35:46Z",
  "coordinates_obscured": false,
  "observation_field_values": […]
  "project_observations": […]
  "observation_photos ": […]
  "comments": []
  "taxon": […]
  "identifications": […]
}
```


Example 20: Natusfera observation data for ID 313411 encoded in JSON format (Source: *https://natusfera.gbif.es/observations/313411.json*)

5.3.1 Schema mapping from Natusfera to STAplus

To conceptualize how class attributes in the STAplus data model (see Figure 1) are mapped from the Natusfera portal, the following diagrams demonstrate schema mapping by each class for the sake of simplicity.

First of all, establishing Sensor class to monitor biodiversity is not common as the majority of users on the Natusfera portal use their individual camera to upload pictures of some species as well as other users identify the species name based on the uploaded pictures. In this case, since the model type of camera is not required information in Natusfera, the name of Sensor class can be 'generic camera', or even 'human eye' for the purpose of identification by other users. The users registered in Natusfera have their login name recorded as 'user_login' or 'identifications[…].user.login', therefore can refer to 'name' in Party class (see Figure 8).

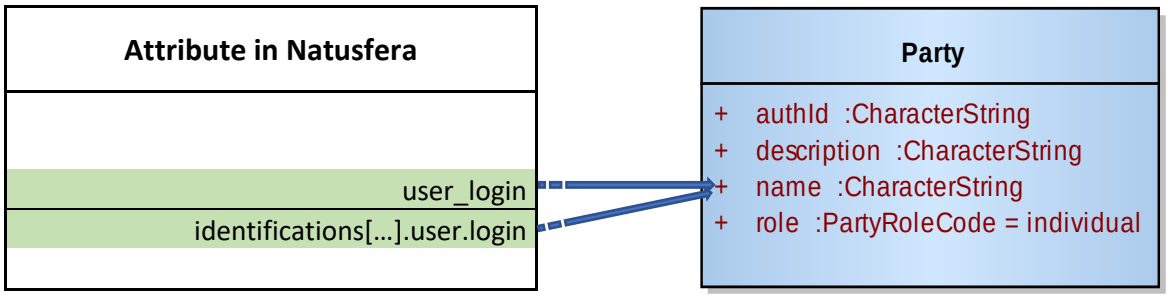

Figure 8: Schema mapping for Party

Although the name in Thing class can be a camera of some user (see Figure 9), its associated Location (or Historical location) class does not have a static station, which may not be applicable to biodiversity domain in general. Where each observation was recorded using a camera owned by some user (Party) is rather relevant to species identification and distribution.

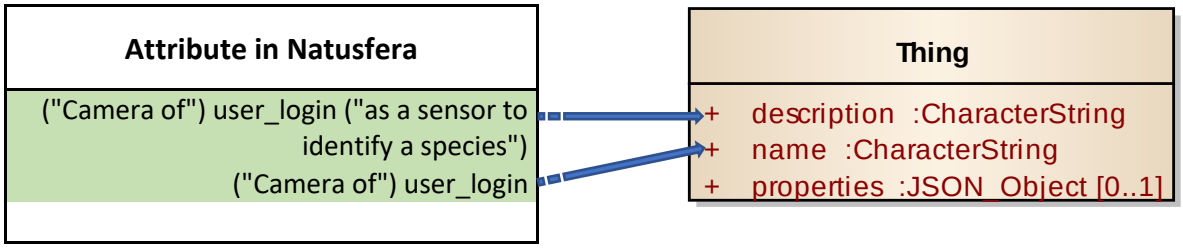

Figure 9: Schema mapping for Thing

Various parameters associated with Observation class can be mapped from attributes in Natusfera (see Figure 10) including 'species_guess' and 'quality_grade' as well as the name of place, the number of individuals and its sex. The time observed at a specific time zone ('time_observed_at_utc') can be transformed to 'phenomenonTime' while the time of record created ('created_at_utc') refers to 'resultTime'. The standardized taxonomy ID ('taxon_id') classified by the GBIF Backbone Taxonomy (https://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c) can be useful information to refer to 'result' while 'observation_photos[…].photo_id' being mapped to 'result' as visual aid for species identification.

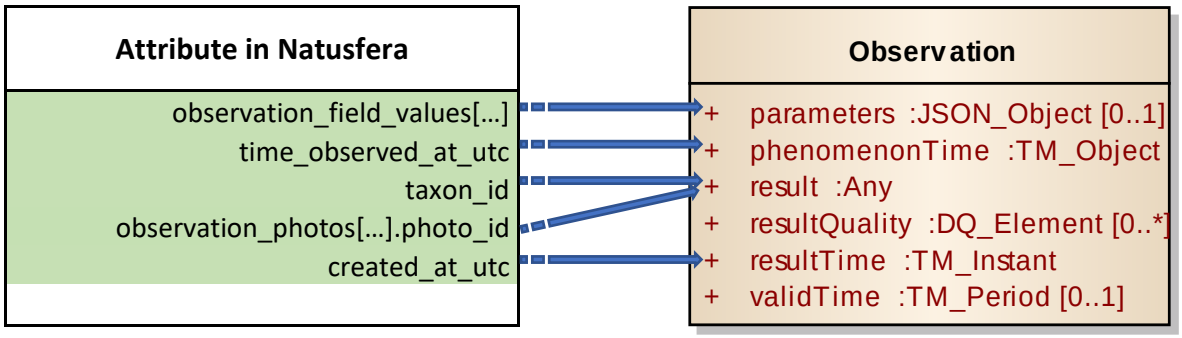

Figure 10: Schema mapping for Observation

The location data is fundamental for FeatureOfInterest class associated with Observation class, which can be expressed in geographical coordinates such as 'latitude' and 'longitude' (see Figure 11). In addition, descriptive attributes in Natusfera ('positional_accuracy', 'positioning_device', 'positioning_method', 'coordinates_obscured') may refer to 'description' as well as 'place_guess' being 'name'.

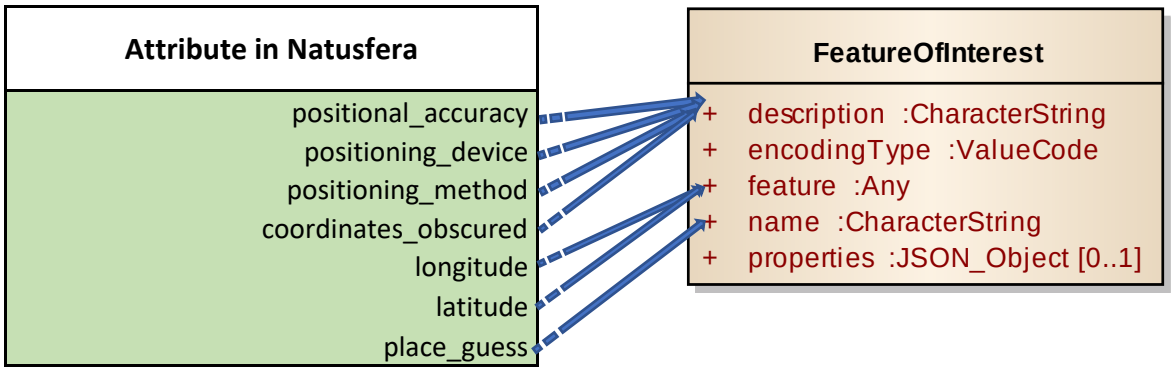

Figure 11: Schema mapping for FeatureOfInterest

In Natusfera a collection of Observations may belong to Group owned by specific Party. As shown in Figure 12, the attribute 'creationTime' in Group class can be derived from the first

observation record created ('created_at_utc') as well as 'description' can be the scientific name of species identified in taxonomy ('species_guess'). Since the 'uri' (http://natusfera.gbif.es/observations/313411) contains the observation group ID (e.g. 313411), it may simply refer to 'name' in Group class.

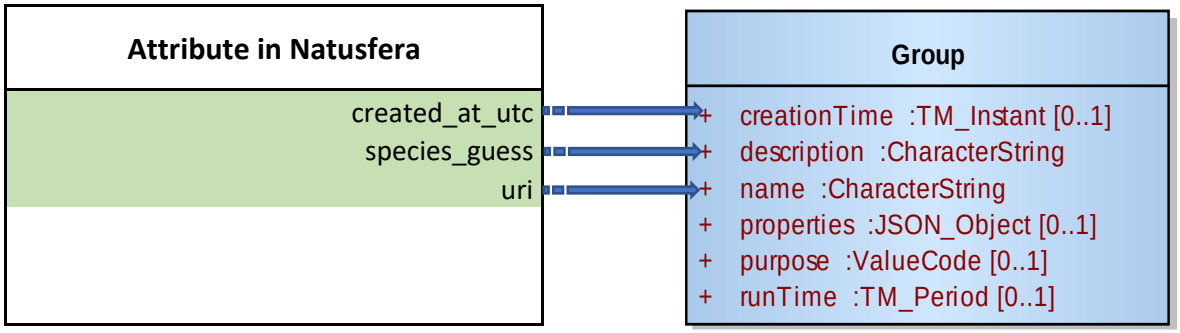

Figure 12: Schema mapping for Group

As Datastream class is specific to each Party as an owner, 'user_login' shall be mapped to 'name' in Datastream class (see Figure 13). On the other hand, 'description' in Datastream class may be referred by as many Natusfera attributes as preferred including project ID ('project_observations[…].project.id'), project title ('project_observations[…].project.title'), and license ('license'), deriving from Project class (see Figure 14) and License class (see Figure 15). The Natusfera attribute 'projects.created_at' can simply refer to 'creationTime' specific to Project class as well as 'project.terms' being mapped to 'termsOfUse'. The 'uri' in Project class may be transformed from the uri https://natusfera.gbif.es/projects/ followed by the Natusfera attribute 'project_observations[…].project.title'.

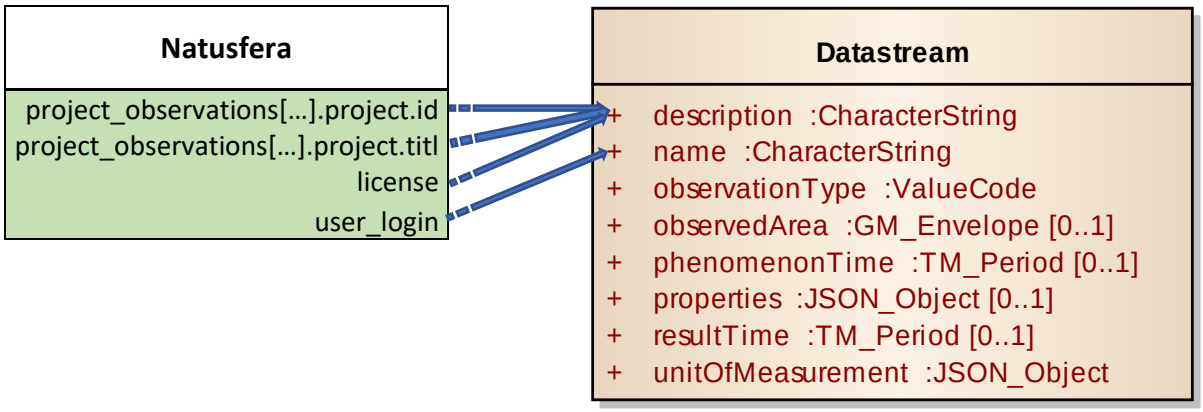

Figure 13: Schema mapping for Datastream

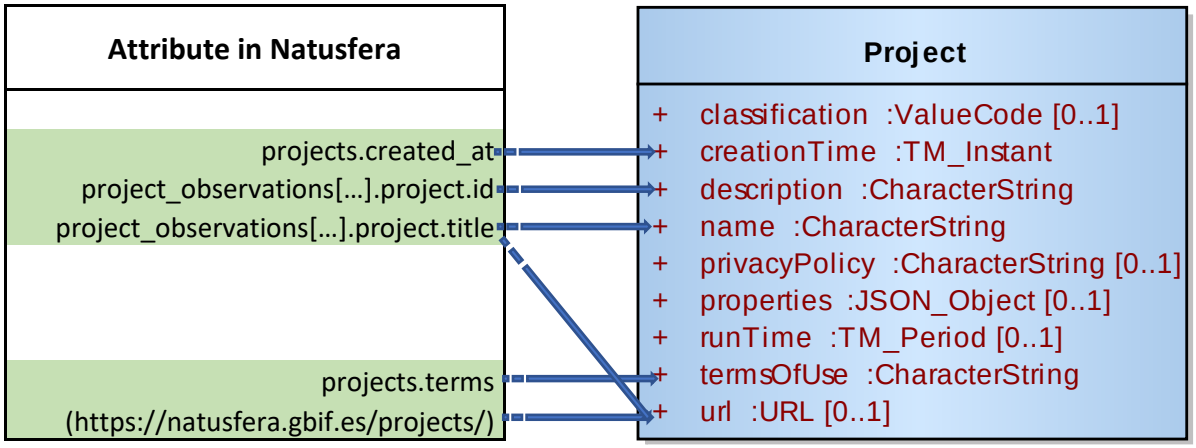

Figure 14: Schema mapping for Project

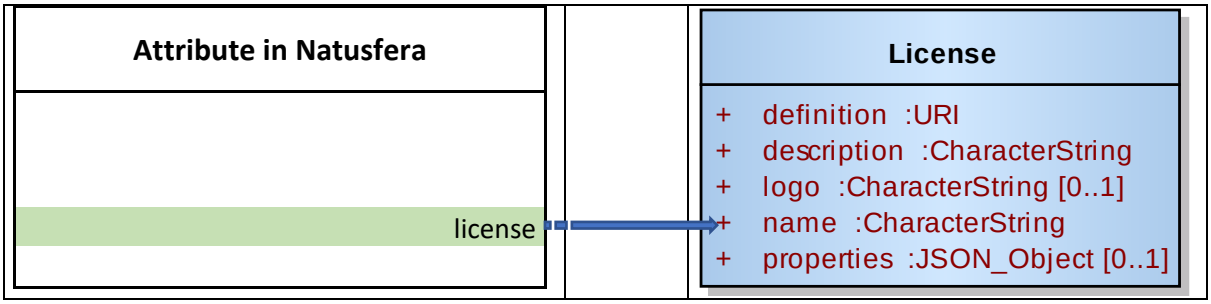

Figure 15: Schema mapping for License

5.3.2 Testing roles of Relation class with Darwin Core terms

Following schema mapping to export Natusfera records into the STAplus deployed in the EGI (EOSC) cloud, relations were created to describe various roles of relations, defined by Darwin Core terms (dwc:), between classes Observation, Group and Party. The examples stored in the cloud (https://cos4cloud.secd.eu/sta4cs/v1.1) are as follows:

- dwc: recordedBy (http://rs.tdwg.org/dwc/terms/recordedBy)
  - Example 21 where Subject = Observation (photo), externalObject = Party
- dwc: identifiedBy (http://rs.tdwg.org/dwc/terms/identifiedBy)
  - Example 22 where Subject = Observation (identification), externalObject = Party
- dwc: inCollection (http://rs.tdwg.org/dwc/iri/inCollection)
  - Example 23 where Subject = Observation (photo), externalObject = Group
  - Example 24 where Subject = Observation (identification), externalObject = Group
- dwc: toTaxon (http://rs.tdwg.org/dwc/iri/toTaxon)
  - Example 25 where Subject = Observation (photo), Object = Observation (identification)

```
{
"Subject": {"@iot.id": 1},
"externalObject":"https://cos4cloud.secd.eu/sta4cs/v1.1/Parties('a00d3f14-a085-38cf-86a0-e234b9d5b84c')",
"description":"recordedBy",
"role":"http://rs.tdwg.org/dwc/terms/recordedBy"
}
```
Example 21: Create a relation with dwc: recordedBy

```
{
"Subject": {"@iot.id": 2},
"externalObject":"https://cos4cloud.secd.eu/sta4cs/v1.1/Parties('a00d3f14-a085-38cf-86a0-
e234b9d5b84c')",
"description":"identifiedBy",
"role":"http://rs.tdwg.org/dwc/terms/identifiedBy"
}
```

Example 22: Create a relation with dwc: identifiedBy

```
{
"Subject": {"@iot.id": 1},
"externalObject":"https://cos4cloud.secd.eu/sta4cs/v1.1/Groups(1)",
"description":"inCollection",
"role":"http://rs.tdwg.org/dwc/iri/inCollection"
}
```

Example 23: Create a relation with dwc: inCollection where observation is photo

```
{
"Subject": {"@iot.id": 2},
"externalObject":"https://cos4cloud.secd.eu/sta4cs/v1.1/Groups(1)",
"description":"inCollection",
"role":"http://rs.tdwg.org/dwc/iri/inCollection"
}
```

Example 24: Create a relation with dwc: inCollection where observation is identification

```
{
"Subject": {"@iot.id": 1},
"Object": {"@iot.id": 2},
"description":"toTaxon",
"role":"http://rs.tdwg.org/dwc/iri/toTaxon"
}
```

Example 25: Create a relation with dwc: toTaxon

5.4 Best Practice when importing Natusfera/iNaturalist into STAplus

Section 3.3 describes a mapping between the Natusfera/iNaturalist model. In this section we present a JavaScript implementation for reading selected Natusfera/iNaturalist records and creating the equivalent information in a STAplus server. Both Natusfera and STAplus use JSON as a media type to communicate the data. We have selected JavaScript to implement it due to the native capabilities that JavaScript offers for processing JSON information and the capability to execute asynchronous calls. We used an HTML page as a user interface that can be executed in a web browser (Figure 16) but we suspect other JavaScript environments such as node.js could also execute the routine.

The main function NatRecordList2STA() receives a STAplus URL where the data will be stored and a list of Natusfera/iNaturalist URLs to observations, and then requests them one by one (e.g. https://natusfera.gbif.es/observations/313411.json) that will be transformed into STAplus objects. The code executes a sequence of functions that extracts the necessary information from the Natusfera observations and verifies if the equivalent object in the

STAplus model exists. If it exists, the code saves the ID of the object; if it does not exist, it creates it and recovers the ID from the Location header of the HTTP POST response. The IDs recovered will be used to create the relations between the objects while creating new ones. The code starts with the "leaf" objects in the STAplus model: Party, Project, Sensor, ObservedProperty, License, `Thing`, FeatureOfInterest and Group. Then it continues with the objects that will require the IDs of the previous objects to be created if needed: Datastream, Observation. It repeats the same sequence for the "taxon" observation (that is associated to a "human sensor", for the "picture" observations (that are associated to a "camera sensor") and the "taxon validation" observations (that are also associated to a "human sensor"). Pictures are not duplicated and are saved as a link to the original version in Natusfera/iNaturalist.

The sequence is completely automatic and the result of each individual object request is shown in a text area in the HTML interface indicating the object ID and if has been created or was already present in the STAplus server. When the creation process is finalized, a STAplus request for the created group is built and offered to the user as a link in the HTML page for verification purposes.

The way the code is presented in the HTML page implies that a person interested in populating a STAplus server can use it to transform a relatively small number of Natusfera/iNaturalist observations. In the future we plan to extend the development to support a Natusfera/iNaturalist search result as an input. This way a user will be able to formulate a query for the information that one is interested in and save the result in a STAplus service for further research using the capacities of the ODATA queries that go beyond the Natusfera/iNaturalist query syntax. In principle the code could be used to import a large number of records but it was not the initial purpose. For massive imports the code should be modified to reduce the number of requests to the STAplus service by sending and creating more than one object at once.

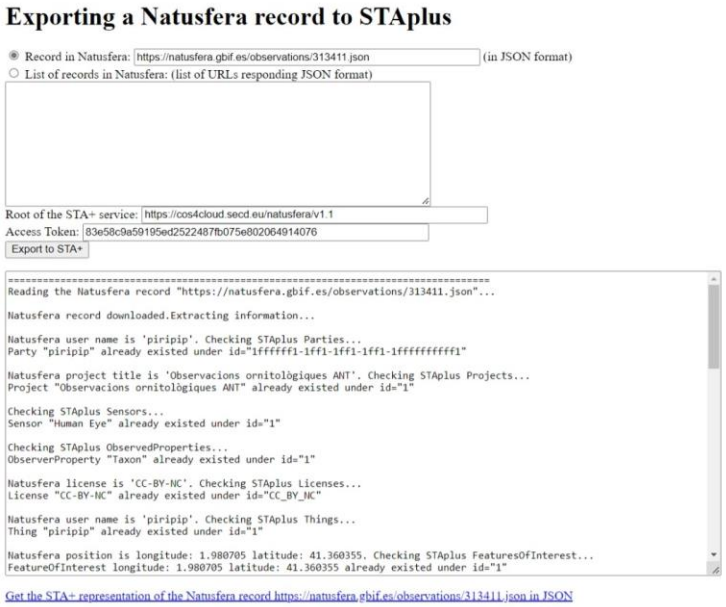

Figure 16: User interface of the Natusfera/iNaturalist to STAplus routine

5.5 Best Practice when modelling Pl@ntNet as a Service

"Pl@ntNet data as a Service" aims at publishing the current 10+ million Pl@ntNet (PN) observations dataset through a STAplus API. PN observations are mapped to SensorThings format through a custom data loader: https://github.com/plantnet/staplus-data-loader/. This mapping heavily relies on concepts introduced by STAplus such as "observations bags" and "relations".

Figure 17 illustrates a Pl@ntNet observation of a *Meconopsis cambrica (L.) Vig.* (Welsh Poppy) by Hervé Goëau. This observation has 5 pictures (2 leaves, 2 flowers, 1 fruit), was produced on May 28, 2014 and is geolocated at Giverny, France. License is CC BY-SA.

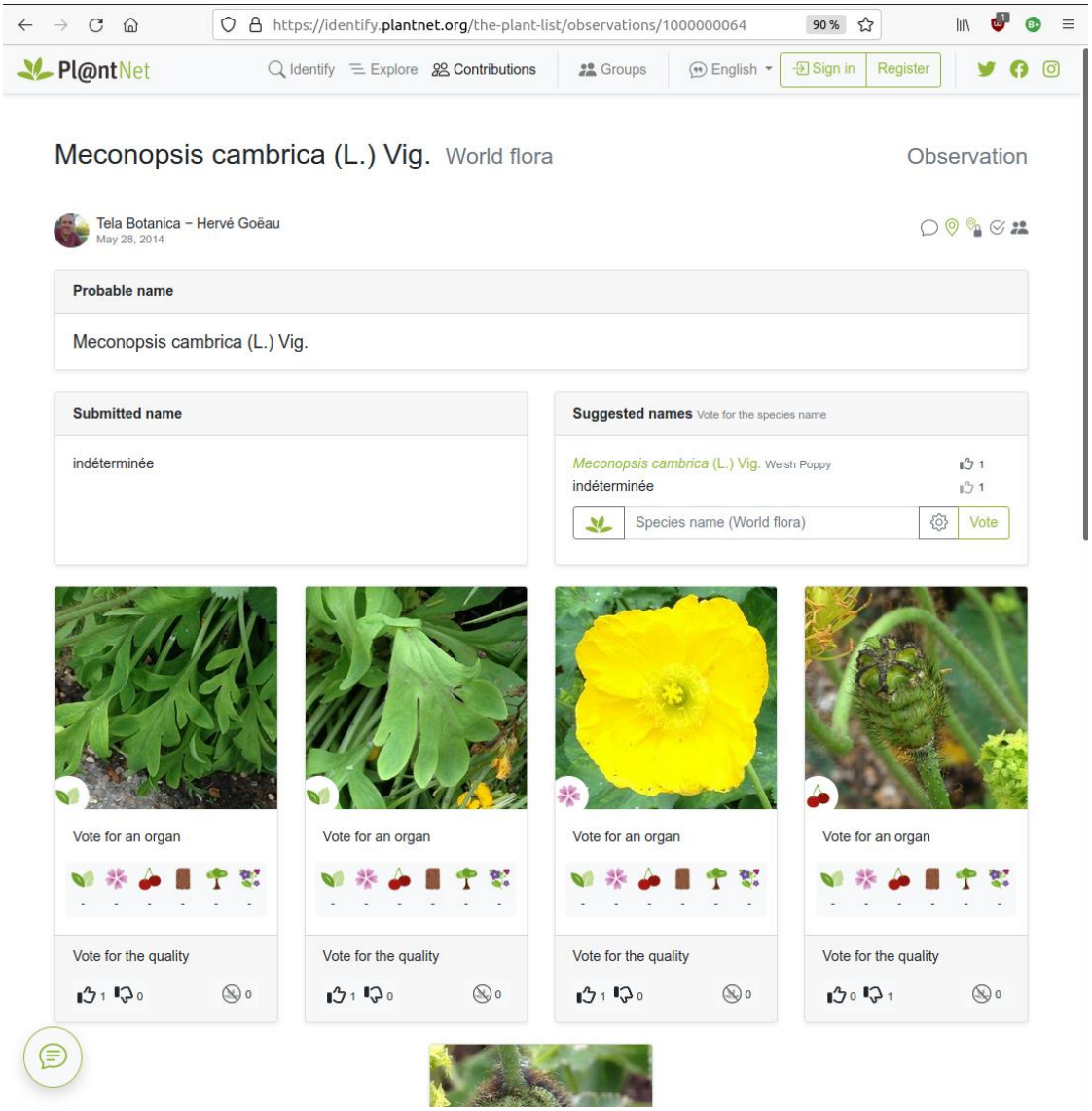

Figure 17: Pl@ntNet observation (ID 1000000064): https://identify.plantnet.org/weurope/observations/1000000064


```
{
    "id": "1000000064",
    "author": {
        "id": "100198361",
        "name": "Tela Botanica – Hervé Goëau"
```

```
    },
    "dateObs": "May 29, 2014",
    "dateUpdated": "Mar 5, 2022",
    "license": "cc-by-sa",
    "currentName": "Meconopsis cambrica (L.) Vig.",
    "submittedName": "indéterminée",
    "images": [
        {
            "id": "50f599eae29739aa3dc4b848fa43e9987bf53a36",
            "o": "https://bs.plantnet.org/image/o/50f599eae29739aa3dc4b848fa43e9987bf53a36",
            "organ": "leaf"
        },
        {
            "id": "628c7c2387e02b797b4c6d2dfc0fd86f8e654b4f",
            "o": "https://bs.plantnet.org/image/o/628c7c2387e02b797b4c6d2dfc0fd86f8e654b4f",
            "organ": "leaf"
        },
        {
            "id": "1523ad3e444758d57f92fa61cb16998b216141a5",
            "o": "https://bs.plantnet.org/image/o/1523ad3e444758d57f92fa61cb16998b216141a5",
            "organ": "flower"
        },
        {
            "id": "3e8b8750102fed86e8461652ba5061c55c00ff15",
            "o": "https://bs.plantnet.org/image/o/3e8b8750102fed86e8461652ba5061c55c00ff15",
            "organ": "fruit"
        },
        {
            "id": "548ab2d7e12dfdd9925908888765955f917e751c",
            "o": "https://bs.plantnet.org/image/o/548ab2d7e12dfdd9925908888765955f917e751c",
            "organ": "flower"
        }
    ],
    "isValid": false,
    "votes": {
        "determinations": [
            {
                "value": "Meconopsis cambrica (L.) Vig.",
                "species": {
                    "name": "Meconopsis cambrica",
                    "author": "(L.) Vig.",
                    "commonNames": ["Welsh Poppy"],
                    "project": "cevennes"
                },
                "name": null,
                "count": 1,
                "selected": false
            },
            {
                "value": "indéterminée",
                "species": null,
                "name": "indéterminée",
                "count": 1,
                "selected": false
            }
        ],
        "malformed": {
            "count": 0,
            "selected": false
        }
    },
    "hasFeedbacks": false,
    "dateObsTSms": 1401314400000,
    "dateUpdatedTSms": 1646513935909,
    "licenseUrl": "https://creativecommons.org/licenses/by-sa/4.0/"
}
```


Example 26: Pl@ntNet native JSON format is pretty straightforward; it contains human-readable public information about the observation

The observation becomes a group, non-mapped information is stored under .properties; FeatureOfInterest is the geolocated plant individual; determination, pictures and organs are observations in their respective user(Party)-specific datastream.

```
https://thymerais.cirad.fr/cos4cloud/api-demo/v1.1/Groups(1)?
  $expand=
    Relations,
    Observations(
      $expand=
        FeatureOfInterest,
        Datastream($expand=Project,Party,License,Thing,Sensor))
```

Example 27: Pl@ntNet observation data mapped to STAplus format

Data mapping highlights Important notes:

- at the time of submission, STAplus and FROST-Server versions used are not the latest. The data model mapping is therefore subject to change in a future iteration,
- relations between observations are not yet using any ontology standard; Darwin Core is likely to be used,
- this use case does not yet describe best practices about how to alter a PN observation across time (add votes from other parties, delete observation, make determination evolve according to votes consensus…).

Three objects describing the three types of observations produced by Pl@ntNet application are reused for all Observations:

- picture: the image produced by the camera,
- organ: the plant organ submitted by the user before identification,
- determination: the most probable Latin name returned by AI identification system.

STAplus Data Model Class	Description
Sensors	2 objects are reused for all Observations: “Generic camera” (picture) and “Pl@ntNet (determination).
Parties	Pl@ntNet user ID is stored as "authId" for easier queries / updates.
Things	Only 1 Thing is created for every Party: “Generic device of party @iot.id…”.
Datastreams	3 Datastreams are created for every Party: “Pictures”, “Taxons” (i.e. determinations), “Organs”.
FeaturesOfInterest	Location of the plant observed. Geolocation is stored as "feature.coordinates". Additional data such as geolocation accuracy or altitude, is stored as "properties".
Observations	For determining Observations, additional taxonomic data such as genus, family or synonyms, is stored as "parameters".
Groups	Represents a whole Pl@ntNet plant observation. Additional data that might be relevant to service consumers, such as original determination or validity state, is stored as “properties".
MultiDatastreams	Not used.

Table 1: STAplus Classes and their use with Pl@ntNet

5.6 Best Practice on how to visualize STAplus data in a map

To create a map from a STA server, we need to focus on positions. Positions can be found in two places in the STA model: The location and the feature-of-interest. The first will contain the geospatial location of the sensor, while the second should provide information about the position of the sensed object. Many observations are intrusive and situate the sensor inside

the measured phenomenon such as an electronic in-situ air quality detector or a water level gauge. Other observations are done remotely by measuring radiation coming from the phenomenon such us the light of a distant star captured in telescope or a Sentinel 2 remote sensing satellite measurement for optical reflectance coming from Earth surface.

Commonly, while creating a map the intention is to represent the real world so that we will be interested in representing the phenomenon and its position and that we will select the feature-of-interest as our geospatial information source.

5.6.1 STAplus Viewer App by CREAF

Digital maps should be interactive, so apart from representing the features of interest it is important to get some extra information about the observations done, that become properties of the represented objects that can be shown when the user clicks on or hovers over an object. Therefore, all initial requests will be done to the FeatureOfInterest instance: https://cos4cloud.secd.eu/sta4cs/v1.1/FeaturesOfInterest.

By default, a query to a feature-of-interest only provides links to observations. The $expand(Observations) parameter will replace the link by the actual object. However, STA objects have the tendency to be too verbose, providing IDs and other information which is not easy to interpret by the final user. The parameter $select allows us to limit the amount of information retrieved from the observation to the one essential for the user: result and phenomenonTime.

The same logic can be applied to get selected information from another object in the data model such as the unitsOfMeasure in the DataStream, the name of the thing (the platform of the sensor), the name of the party (the citizen), the name of the project (e.g. the campaign name) and the description of the license. This filter should be applied to both datastream and multi-datastream.

Finally, the user looking at a map is only interested in the object inside the bounding box represented by the viewport to apply a spatial filter that selects only the information inside the bounding box will limit the area of the data to the relevant zone. This is done by applying a $filter=st_within(feature,geography'POLYGON(minx miny, maxx miny, maxx maxy, minx maxy, minx miny). Table 2 presents the complete URL decomposed in its relevant parts.

Query fragment	Explanation
https://cos4cloud.secd.eu/sta4cs/v1.1/FeaturesOfInterest?	Starting by asking about *FeaturesOfInterest*
$select=feature,id&	I only want the *feature* and *id* elements in the *FeatureOfInterest*
$expand=Observations(	Do not give a link to *Observations* but the object itself
$select=result,phenomenonTime;	I only want the *result* and *phenomenonTime* elements in the FeatureOfInterest
$expand=	Do not give a link to *DataStream* and *MultiDataStream* but the objects themselves
Datastream(	From the *DataStream*
$select=unitOfMeasurement,name;	I only want the *unitsOfMeasure* and the *name*

Query fragment	Explanation
$expand=	Do not give a link to *Thing, Party, Project, Licence* but the objects themselves
Thing($select=name),	I only want the *name* element in the *Thing*
Party($select=name),	I only want the *name* element in the *Party*
Project($select=name),	I only want the *name* element in the *Project*
License($select=description)),	I only want the *description* element in the *License*
MultiDatastream(	From the *MultiDataStream*
$select=unitOfMeasurements,name;	I only want the *unitsOfMeasure* and the *name*
$expand=	Do not give a link to *Thing, Party, Project, Licence* but the objects themselves
Thing($select=name),	I only want the *name* element in the *Thing*
Party($select=name),	I only want the *name* element in the *Party*
Project($select=name),	I only want the *name* element in the *Project*
License($select=description)))&	I only want the *description* element in the *License*
$filter=st_within(feature,geography'POLYGON((46 13,50 13,50 9,46 9,46 13))')	I do not want all *FeaturesOfInterest* only the ones within a polygon.

Table 2: STAplus query divided in parts and explained

Note: this URL was elaborated thanks to the essential information provided in this page: https://developers.sensorup.com/docs/#introduction

The following JSON fragment represents the response of an HTTP GET request formed using the presented approach and sent to a 52°North STA implementation.

NOTE: 52°North implementation requires that the objects that your request to $expand have been previously selected using a precedent $select (this is not required in the FROST implementation).

```
value:
  0:
    @iot.id:                    "f51f67cf-ced2-4e76-bec9-11f47b78ca81"
    feature:
      type:                     "Point"
      coordinates:
        0:                      6.56473
        1:                      51.3027951
      crs:
        type:                   "name"
        properties:
          name:                 "EPSG:4326"
    Observations:
      0:
        result:                 "https://cos4cloud.sta.52north.org/v1.1/files/1630139197694--2021-06-02_04-08-25-559548_0.mp4"
        phenomenonTime:         "2021-06-02T04:08:27.000Z"
        Datastream:
          name:                 "imagery datastream"
          unitOfMeasurement:
            name:               "n/a"
            symbol:             ""
            definition:         "https://www.merriam-webster.com/dictionary/picture"
          Party:                {}
          Thing:
            name:               "Raspberry Pi 4 B, 4x 1,5 GHz, 4 GB RAM, WLAN, BT"
      1:
        result:                 1016
        phenomenonTime:         "2021-06-03T09:02:57.000Z"
        Datastream:
          name:                 "Pressure"
          unitOfMeasurement:
```


Example 28: Response of the STAplus query

The actual HTTP GET request URL used in this case to get the previous response was:

```
https://cos4cloud.sta.52north.org/v1.1/FeaturesOfInterest?
  $select=feature,id,Observations&
    $expand=Observations(
      $select=result,phenomenonTime,Datastream;
        $expand=Datastream(
          $select=unitOfMeasurement,name,Thing,Party;
          $expand=Thing($select=name),Party($select=name)))
```

Example 29: Nested request of the STAplus query

The response of a query like this is quite similar to what will be expected from a geospatial service such as an OGC API Features. It is not difficult to adapt a client that is able to parse a GeoJSON file to also parse a response like this and extract the "geometry" from the "feature" object and the "properties" from the relevant attributes of the observations object array.

To demonstrate this possibility, we adapted the MiraMon Map Browser in this direction and we were able to create representations like the one presented in the following illustrations where observations of some experimental camera traps are presented to the user as point in a map that can be queried to see the observations that include environmental observations in combination with pictures of the suggested or identified animal, captured by the camera.

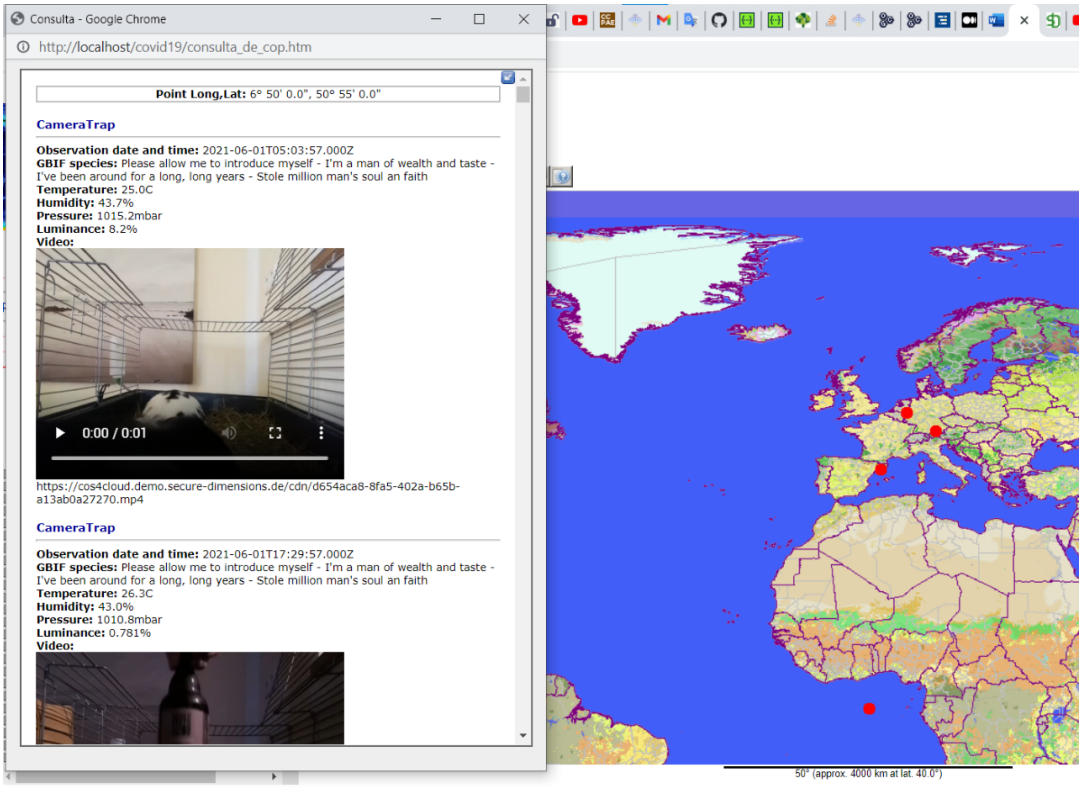

Figure 18: STAplus response shown in the interactive MiraMon Map Browser

5.6.2 STAplus Viewer App by Secure Dimensions

The STAplus Viewer App is a proof of concept implementation as Web-Browser application based on JavaScript and Leaflet. The implementation further leverages JS libraries from STAM (SensorThings API Map) developed by Datacove for INSPIRE: https://github.com/DataCoveEU/API4INSPIRE

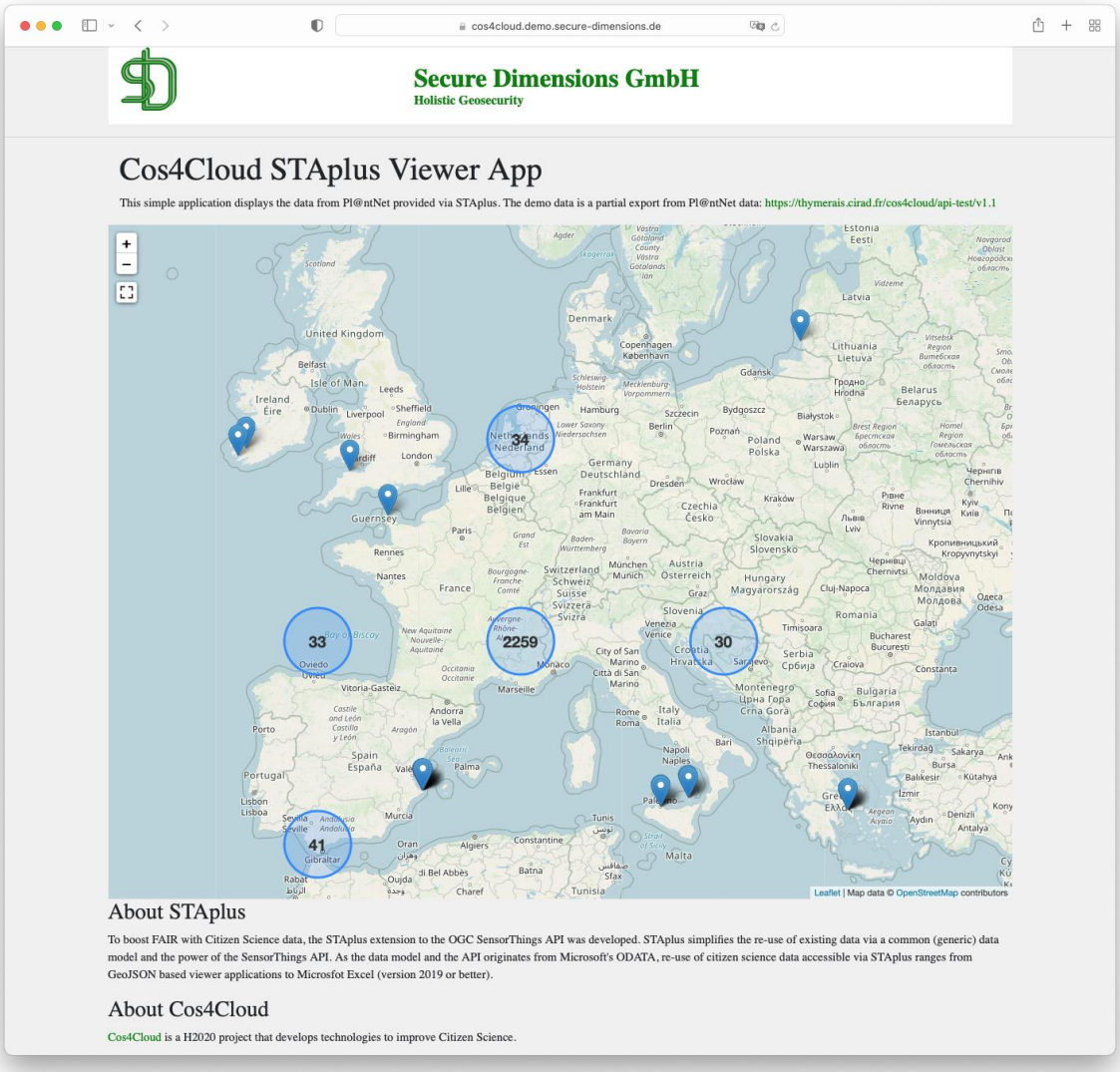

Figure 19: STAplus Viewer App (https://cos4cloud.secd.eu/staplus-viewer-app) showing Pl@ntNet data

The good thing about using standards is that existing libraries work out of the box. The configuration is simple to load all locations via the FeaturesOfInterest. The following kind of default configuration for the STAM.queryObject already works:

```
queryObject: {
    count: true,
    skip: 0,
    entityType: 'FeaturesOfInterest',
    filter: null,
    select: null,
    expand: null,
    top: 0
}
```

Example 30: STAM queryObject configuration

However, the result contains the Observations for each feature-of-interest. Because the observations are modelled via STAplus groups, we are only interested in the locations of the

feature-of-interest and their number. To keep the response small and gain better performance, we modified the default expand from STAM:

```
queryObject: {
    count: true,
    entityType: 'FeaturesOfInterest',
    filter: null,
    select: null,
    top: 0
}
```

Example 31: STAM queryObject returning count for feature-of-interest

Adapting the Viewer App to display data from STAplus was mainly achieved by creating a specific STAM.markerClick function.

```
markerClick: function (foi) {
    var div = document.createElement('div');
    div.id = 'result';
    div.style = 'overflow-y: scroll; height:800px;';
    div.innerHTML = '<h3>' + foi.properties.description + '</h3>';
    makeRequest(foi.properties[ '@iot.id']);
    return div;
}
```

Example 32: STAM markerClick example

The STAplus specific fetching of the observations, respecting the group modelling, is implemented within the ‘makeRequest’ function. The gimmick comes with constructing the request URL. The following example illustrates a nested expand/select to gain good performance:

```
https://thymerais.cirad.fr/cos4cloud/api-test/v1.1/Groups?
    $filter=Observations/FeatureOfInterest/@iot.id eq foiId&
    $expand=Observations(
        $expand=Datastream(
            $select=unitOfMeasurement;
            $expand=
               License($select=name),
               Party($select=name)
        )
    )
```

Example 33: STAM markerClick STAplus specific query via the Group class

In essence, the ‘makeRequest’ function uses the XMLHttpRequest object to fetch the group’s observations where the FeatureOfInterest/@iot.id is equal to the id of the FeatureOfInterest instance that is linked to the marker.
The XMLHttpRequest uses a callback on ‘readystatechange’ to render the JSON encoded response from the STAplus endpoint into HTML. This demo application uses a simple HTML list format as illustrated below:

Figure 20: STAplus Viewer App showing Pl@ntNet the result of querying by location

The STAplus Viewer App is based on the Leaflet JavaScript library. It is however possible to base the application on OpenLayers. This alternative implementation is available from this URL: https://cos4cloud.secd.eu/staplus-viewer-app/ol.html.

Even though it looks and feels slightly different, both implementations provide the same functionality. The difference of instantiation is illustrated in Table 3:

Leaflet based

```
var mymap =
L.map('mapid').setView([47.997791,
7.842609], 5);
mymap.addControl(new
L.Control.Fullscreen());
L.tileLayer('https://{s}.tile.iosb.fraunhof
er.de/tiles/osmde/{z}/{x}/{y}.png', {
    attribution: 'Map data &copy; <a
href="https://www.openstreetmap.org/">OpenS
treetMap</a> contributors',
    maxZoom: 25
}).addTo(mymap);
L.stam({
    baseUrl:
"https://thymerais.cirad.fr/cos4cloud/api-
test/v1.1",
    MarkerStyle: "yellow",
    clusterMin: 20,
    queryObject: {
        count: true,
        skip: 0,
        entityType: 'FeaturesOfInterest',
        filter: null,
        select: null,
        expand:[ {
            entityType: 'Observations',
            select:[ 'phenomenonTime'],
            top: 1,
            count: true
        }],
        top: 0
    },
    markerClick: function (foi) {
        //console.log("FoI.id: " +
foi.properties[ '@iot.id']);
        var div =
document.createElement('div');
        div.id = 'result';
        div.style = 'overflow-y: scroll;
height:800px;';
        div.innerHTML = '<h3>' +
foi.properties.description + '</h3>';
        makeRequest(foi.properties[
'@iot.id']);
        return div;
    }
}).addTo(mymap);
```

OpenLayers based

```
var map = new ol.Map({
    renderers:[ 'Canvas', 'VML'],
    layers:[
    new ol.layer.Tile({
        source: new ol.source.OSM(),
    }),],
    target: 'mapid',
    view: new ol.View({
        center:[808701.59, 6493626.85],
        zoom: 5,
    }),
});
var sta = new ol.layer.STAM({
    map: map,
    baseUrl:
"https://thymerais.cirad.fr/cos4cloud/api-
test/v1.1",
    MarkerStyle: "yellow",
    clusterMin: 20,
    queryObject: {
        count: true,
        skip: 0,
        entityType: 'FeaturesOfInterest',
        filter: null,
        select: null,
        expand:[ {
            entityType: 'Observations',
            select:[ 'phenomenonTime'],
            top: 1,
            count: true
        }],
        top: 0
    },
    markerClick: function (foi) {
        console.log("FoI.id: " +
foi.properties[ '@iot.id']);
        var div =
document.createElement('div');
        div.id = 'result';
        div.style = 'overflow-y: scroll;
height:800px;';
        div.innerHTML = '<h3>' +
foi.properties.description + '</h3>';
        makeRequest(foi.properties[
'@iot.id']);
        return div.outerHTML;
    }
});
map.addLayer(sta);
```

Table 3: Comparing the code used in Leaflet and in OpenLayers

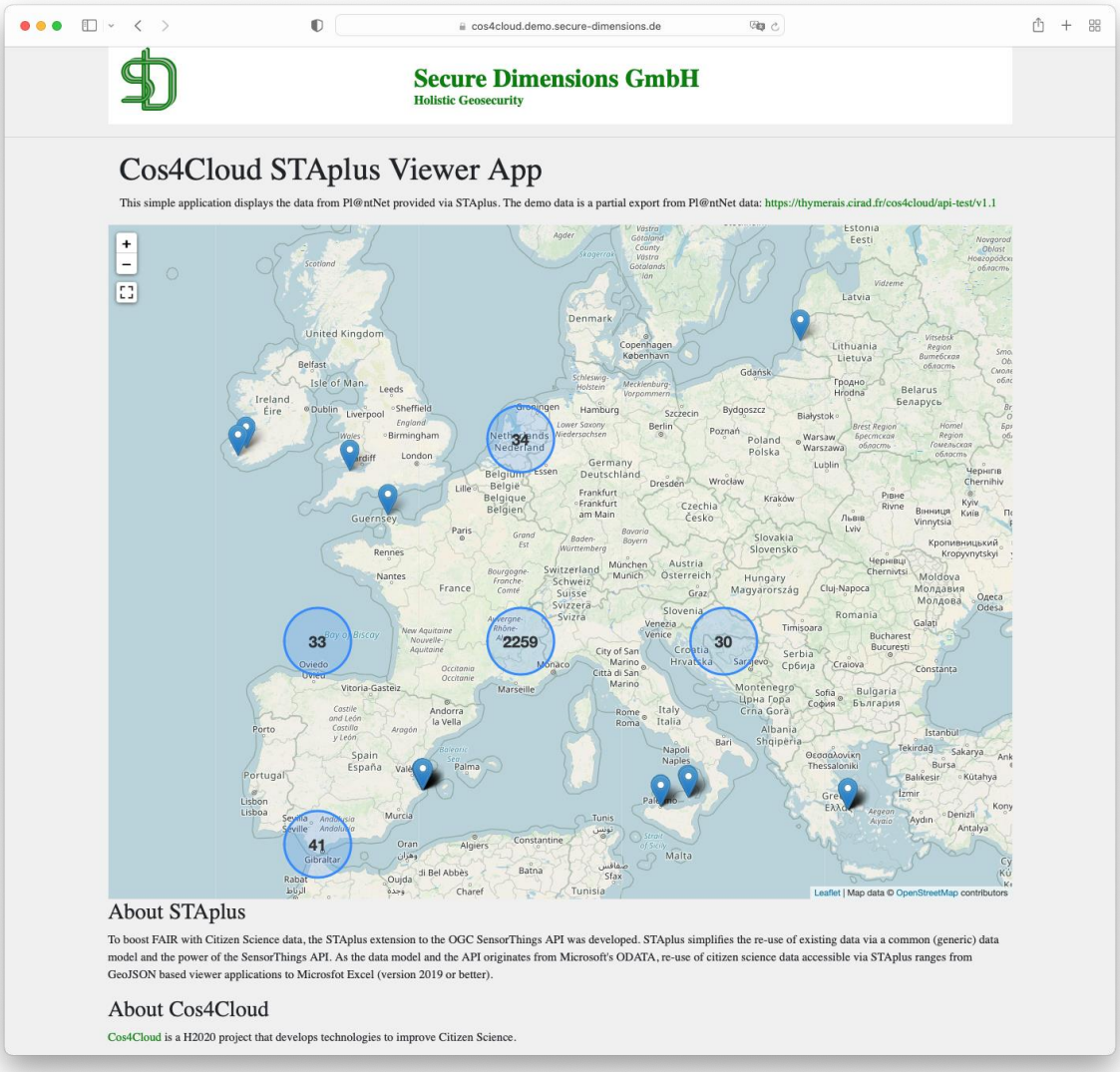

Figure 21: OpenLayers based implementation of the STAplus Viewer App (https://cos4cloud.secd.eu/staplus-viewer-app/ol.html)

5.7 Best Practice modelling community processes

We will describe some real-world appliances of how the dynamic potential of the data model could be unfolded and what aspects a Citizen Observatory (CO) has to consider when doing so. The following use case diagram depicts a few workflows which are described next:

- A contributor uploads some observations about an occurrence of an animal
- Contributors enrich data in a particular community process
- A scientist assembles data as a referenceable data collection
- An autonomous, user detached task regularly runs a probability check

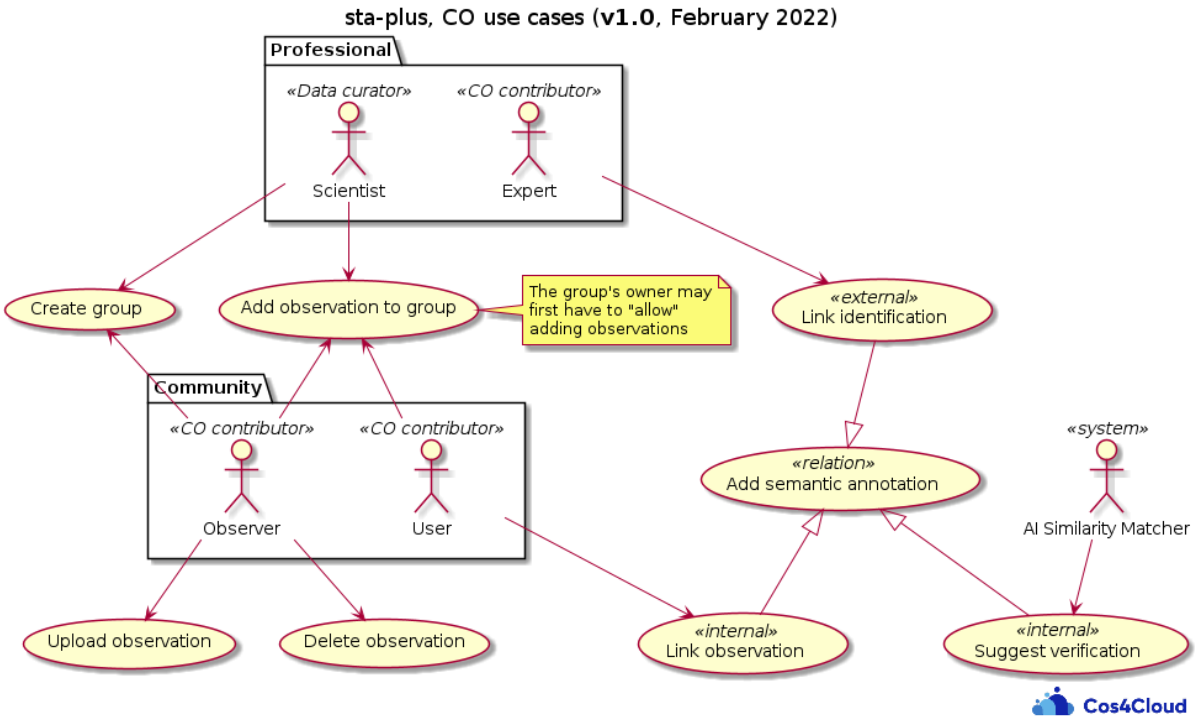

Figure 22: STAplus Citizen Observatories use cases

5.7.1 A contributor uploads some observations about an occurrence of an animal

Situation

In the context of citizen science an observation often is not an isolated thing. For example, the occurrence of an animal or a plant is “proven” by taking a photo or even described in more detail by providing additional data. Such data may also be important and should be modeled explicitly. Saving it in a non-structured “properties” object would make it quite hard to query on the accompanying characteristics of the actual observation made.

Case Descriptions

A member of the CO notices an occurrence of an animal. She takes a picture of it and makes a guess on its species. Both observations are highly linked together, so the CO lets her upload them in a group by selecting a purpose “observation event”. This detail is part of the actual upload process and transparent to the user. However, the platform allows the user to add further observations to that group later on. After uploading a new group is created which includes all observation items made.

Observation items or statements about phenophase or environmental phenomena like temperature or humidity, can be added later. This gives more insight in the observed situation.

The observation event is uploaded on the CO platform now. It depends on the platform to make it available immediately, or let the uploader decide to publish it later on.

Recommended Practice

- Individual observations belonging to one occurrence should be grouped to have one entity which references observations for a given purpose.

- The group concept is generic by intent to support multiple use cases. Per se, it could contain arbitrary observations. A CO may utilize the concept for dedicated purposes. It should maintain a list of purposes which make sense in its application domain.

5.7.2 Contributors enrich data in a particular community process

Situation
Once observation groups are published (see section 5.7.1), they become visible to other members of the CO. The platform provides tools to search and explore observation events on the platform itself. Depending on the tools being used, the community can interact on that observation group. However, the platform could also have mechanisms to notify members (e.g. experts or scientists) who have subscribed for a given location or species they are interested in.

Case Descriptions
The CO platform operates an "observation blog" where users can upload new blog items (the observation groups). A member of the CO finds the new blog item and wants to add the phenophase detected from the photo and the date when the observation took place. The original observer gets notified if she wants to accept the new contribution to be part of her blog item.

An expert is interested in a particular species. He has subscribed to species occurrences in a particular region and takes a closer look. The species guess done here seems to be wrong and he adds a correction. The correction will add a new observation which is then linked as a relation to the group. The relation is being added without explicit acceptance, as the original observer has configured to apply contributions from experts without confirmation.

After correction, another expert reviews the blog item and adds a new relation which links the correct observation to a GBIF species definition by using the "dwc:identifiedBy" predicate. Now, other users have a link to an external database providing more detailed information like the species' taxonomy, etc.

Recommended practice
- The community should be able to act on the observation, either by adding observations to the group or by adding relations with semantic annotation
- Based on the data model the CO implements kind of a control model under what circumstances contributions from other users are allowed
- Particular roles (for example beginner, advanced, expert) can help to regulate the community process
- The CO should allow to also link external objects via semantic annotation

5.7.3 A Scientist assembles data as a referenceable data collection

Situation
Data collected by citizens have become more and more important within scientific research in the recent years. CO platforms act like data hubs and are taken into account by scientists within their research. An interoperable interface helps a lot to work on data either exported or permanently linked via URL. Furthermore, data taken from "public" sources have to meet certain conditions before they can take a valuable part of a scientific work. Just to mention a

few requirements, the data has to meet some level of trust, must be referenceable and reasonable, and have to suffice certain license conditions.

Case descriptions
A scientist does some work on a particular species. For the work she needs trustworthy data. She does some data research starting from GBIF and performs a query on the CO platform using the GBIF identifiers to find all the related observations available on the platform. As a result, she gets a list of relations associated with an observation. Most of the relations are part of an observation group, which she finds helpful to reason about the genesis of the observation. The added information of all the contributors makes it possible for her to filter out those observations which do not have an agreement on the actual result.

The scientist wants to publish her work in a reproduceable manner. She creates a new group where she adds all the observations relevant to her research, adds a proper license and gives the group a speaking purpose indicating the intent for what the collection is created for. The CO makes sure~~,~~ that application specific business rules apply accordingly, for example:

- All original observers get notified (if a proper consensus exists) when their observations get part of a derived work.
- The license has to be compatible with all those licenses referenced by the grouped observations. Each of those licenses can be obtained by the observation's Datastream. The CO enforces all rules stated in the terms-of-use document which clarifies under what circumstances such a data assembly is allowed.
- A mechanism to seal the data assembly, either by copying the data into a read-only database, sealing the original observations to protect against future change, etc.

The created data collection is now protected against change, resolvable via permanent link, and can now be referenced in a scientific document. The data does not have to be part of the publication and can be used to reproduce the results. In addition, all members can review a list of all data assemblies they contributed observations to.

Recommended practice
- The CO makes a clear statement under what conditions observations can be referenced externally (availability guaranty, license, on account deletion, …).
- The CO implements a GDPR compliant way to un-relate user contributions from observations, at least once they are marked as referenced
- The CO implements certain flows to enforce conditions stated in terms-of-use, license compatibility and contributor's ownership
- The CO has to make sure that such data collections are immutable. This could be done by copying the requested status in a read-only database or disallowing future change.

5.7.4 Autonomous, user detached task regularly runs a probability check

Situation
Many contributors are not experts but still their uploads are valuable observations. Once uploaded, the community can discuss and contribute to individual observations, for example, by correcting it or by adding or linking information to it. The community process behind the actual result, however, also provides valuable data. Machine Learning models could be trained on the process to support the community in the process itself.

This community process may take a short time when it is performed in an active project. But this is not necessarily true. Data uploaded not associated to a project context, or existing older data may lack focus of the community. Autonomous tasks could run regularly to check.

Case description
Based on the observations available at the CO platform, a system job is running regularly to match new observation uploads against a probability check. The check is based on a trained machine learning model which took into account what kind of species were detected where and when. When probability goes below a given threshold, the platform informs the user either during the upload process, or after uploading by adding a relation to the new observation group containing a verification suggestion. The suggestion is visible to the community and can be accepted or denied. Such a result can be used in turn as input for the machine learning model to overcome over-fitting.

Recommended practice
- Inform the user that there are jobs running in the background doing analysis on all contribution data, make the purpose transparent, and let the user opt-in or opt-out
- If possible, anonymize analyzed data to suffice GDPR regulations

5.7.5 Summary of recommended practices

The previous examples have shown by describing how the STAplus data model leverages realistic use cases where a citizen science community is involved. All use cases imply a certain amount of dynamic based on the interaction of that community. We have seen that implementing such kind of a community process demands to adhere requirements like intellectual property rights, to ask for privacy consent, or to guarantee the immutability of data used in reproducible research.

A platform operator implementing the use cases drawn in the previous examples needs to respond at least to the following questions:

- What boundary conditions arise from a certain workflow/feature?
- Where do I have to involve the user?
- Where do I need the user's consent (revokable or permanent)?
- What implications does the revokable consent have?
- What guarantees do I want to give and what implications do they have?

6. Considerations on the Business Logic

The introduced STAplus Data Model extension supports different use cases and business models. To be operated in a particular context it is essential to implement a particular "fit for purpose" business logic.

This section introduces best practices for implementing a "fit for purpose" business logic in the context of multi-user access including CRUD, motivated upon requirements from operating STAplus as a Citizen Science API.

6.1 Concept of Ownership

Ownership is a fact that attaches exclusive rights and control on a Party instance. Ownership can be gained, transferred, damaged or lost, e.g. by accident or through impersonation attacks. In the context of this document, the proof of ownership is important as well as the ability and the implications to an implementation that supports the transfer of ownership to another party. Any deployment with support for multi-user CRUD must implement business logic to prevent impersonation attacks.

For the STAplus data model, the concept of ownership can be realized by instantiating the class Party and attaching it to a Datastream, Thing or Group class. An implementation that supports the ownership concept and in particular allows the transfer of ownership must be clear about the implications that are introduced by the STAplus data model and API. There could be a section within the service's terms of use which explains in detail all implications to the user if the transfer of ownership is allowed.

Attaching a party to a datastream implies that all directly associated objects with a 0/1..* relationship to datastream have inherited ownership. Therefore, the classes Observation, Sensor, Thing, ObservedProperty and License inherit the ownership via the (Multi)Datastream class.

Expressing ownership to a thing is important as the ownership controls the ability to update the thing's location. However, it must be specified what the implications to a thing are when the ownership on a datastream or multi-datastream is transferred to another party. Will the ownership on the thing remain unchanged? Will the new owner of the datastream or multi-datastream own a copy of the thing (including all historic locations) or will the ownership move to the new party?

Attaching a party to a group controls the rights on the group. Due to the many-to-many relationship between the Group and Relation or Observation classes, ownership inheritance cannot be derived. What shall happen to the rights on a group if the ownership is transferred to another party? As the SensorThings data model does not support provenance, the transfer of ownership would gain exclusive access rights on all observations, relations and the license associated to the group.

Any class that either has direct or inherited ownership (Party, (Multi)Datastream, Observation, Thing, Group) must be managed by associated party representing the authenticated and uniquely identified user. All classes that do neither have direct nor

inherited ownership are Project, Sensor, License, Relation, FeatureOfInterest, ObservedProperty, Location. They can be seen as common or global classes. Instances of these classes can either be managed by parties or the business logic of the STAplus deployment.

Applying the concept of ownership to the STAplus data model and the API can best be understood when "walking thru" an example use case. The following section illustrates the implications to the business logic based on the Camera Trap.

6.1.1 Illustrating the Concept of Ownership for the Camera Trap Use Case

For the Camera Trap use case, it is possible that one person owns the hardware (Raspberry Pi and the sensor board). This person then operates the camera trap and produces observations. The STAplus representation is quite simple as all objects are under control of this one person (e.g. Alice). But how can it model the fact that Alice gives the Camera Trap to Bob who is running the camera trap in his garden for the next two weeks? After that, Alice continues…

Understanding the implications of the Camera Trap use case to the concept of ownership and how to best apply them, we separate the entire process into a setup and a runtime process. For the setup process, the required instances get created with the skeleton for the runtime context. During runtime, the CT-LoaderApp for STAplus uploads observations of trap events by referencing the instances from the skeleton (e.g. the ID of the thing or datastream).

For the setup, a Camera Trap data loader application must create all datastreams that are associated with the sensors deployed on the thing. In particular, the CT-LoaderApp must create a thing which represents the Camera Trap and that is owned by the acting user. This ensures that the data loader can update the thing's location on the user's behalf when the location of the Camera Trap changes. This can happen automatically each time the trap is moved or if the user re-starts the CT-LoaderApp. The ownership on the thing prevents that other users can set fake locations on the thing.

For the common instances such as license, the CT-LoaderApp should re-use existing objects to prevent convolution.

Switching the trap on triggers the CT-LoaderApp to update the thing's location. The loader then starts a 'while(TRUE)' loop to process detection events. For each animal detection event, the loader would then create a group including observations that belong to the event. Additionally, the CT-LoaderApp may generate one or multiple relation(s).

When Alice gives the camera trap to Bob, what are the implications? Being compliant to the concept of ownership, the location of a thing can only be updated by the owning party. Also, the datastream and multi-datastream as well as sensor, observed-property are linked to Alice – so Alice can change these instances. In particular, the thing is linked to Alice. Therefore, it is not possible for Bob to change the thing's location. So, the application has to create a new thing for Bob to be able to set the location. As a consequence, the application must also create new sensors and datastreams and link them to Bob. Essentially, the application's setup has to create a digital twin of the Camera Trap for Bob.

6.2 Security Considerations

The STAplus extension can be used with or without offering a multi-user platform with CRUD access. In the simplest case, a SensorThings or STAplus API is operated in a mode where only the site operator pushes new entities into the repository. Typical examples are sites that provide weather information, water levels or any other "sensor-based" data. In such a case, the access though the API for (anonymous) users is de-facto read-only. Any attempt to use the API for creating, updating or even deleting class instances should be responded with an HTTP status 405 (not implemented / supported). The status of 401 does not make sense as there is no option available to authenticate. A response with an HTTP status 400 would not be correct as the request itself is correct, meaning just the use of the POST, PATCH or DELETE method is not allowed.

When the deployment is setup to support a multi-user interaction as it is required with platforms for Citizen Science, the use of HTTP methods POST, PUT, PATCH and DELETE to create, update and delete data objects must be considered. This naturally requires considering security implications and attacks that must be prevented or mitigated.

A STAplus deployment that offers create, update and delete of any class instance shall require authentication and implement the ownership concept as described in the following section.

6.2.1 STAplus Impersonation Attacks

To properly realize the ownership concept and support multi-user access allowing create and update operations, one possible attack is impersonation. Impersonation takes place if objects, such as things, datastreams, observations, etc. are linked to a user that is not the original creator.

The first example aims at putting legal responsibility to another user. For example, if the attacker would create a photo datastream, upload illegal images and once finished link the datastream to the victim, then without awareness the victim would be associated with the illegal uploads produced by the attacker.

The second example aims at stealing data objects from trusted and professional users. For example, the attacker had bad accreditation when submitting frog observations. To boost the own accreditation, the attacker could modify the `Party` object on a datastream and thereby transfer the associated observations away from the victim and link them to the own party. Perhaps never being detected by the victim, the attacker is now the owner of excellent observations of frogs that wrongly improve his accreditation.

There is, however, another form of impersonation that cannot be detected by an implementation. The attack is quite simple: the attacker downloads a bunch of observations, e.g. frogs from a trusted and accredited user, and then uploads the same or little modified observations using the own authentication. In such a case, the attacker would get credit for excellent frog uploads. Of course, a search on 'frogs' would now return the genuine and the copied observations, so there is a high likelihood that the attacker gets detected. To mitigate such an attack, the business logic should provide a "report fraud" button on observations. This mechanism could be used by users of genuine uploads when detecting stolen, impersonated or violations to licensing conditions.

For the STAplus extension, detectable impersonation attacks can be placed at different classes:

- Class Party: Changing the authId property to a value of another user's identifier results in impersonation. All instances linked to the party would be associated with another user. Another form of impersonation that would affect individual datastream or multi-datastreams and can take place if the datastream or multi-datastream property of the party gets changed. Both approaches cause that the ownership of the linked thing, sensor, observations also change.
- Class (Multi)Datastream: The modification of the party property would cause that all observations, things, sensors and observed properties of that datastream belong to another user. Such modification of the party property of a (Multi)Datastream instance could be used to steal observations (link own party to other user's (Multi)Datastream instance) or to spam (link other user's party to (Multi)Datastream instance containing spam).

6.2.2 Denial of Service

Not particularly specific to STA or STAplus is the prevention of Denial of Service (DoS). The typical protection against upload of huge observations must be controlled.

A specific DoS attack for the SensorThings data model could be based on recursive identification. Similar to recursive DTD expansions in XML, an implementation should detect when the uploaded data contains recursive linking. This is in particular important for the Batch-Processing API.

6.3 General Considerations on Create, Update and Delete Operations

The SensorThings API and therefore the STAplus API allows that a client application uses the HTTP GET, PUT, POST, PATCH and DELETE methods for reading, creating, updating or even deleting instantiated classes.

Security considerations for deleting data objects are out of scope for this document as the allowance / disallowance must be considered for each deployment.

The following sub-sections illustrate some generic security considerations that come with enabling HTTP methods other than GET.

6.3.1 Create Operation

The create operation can be executed by using the HTTP POST method to send the content of the object to be created with the HTTP request's body, using the JSON encoding.

The STA supports different ways on how to create an object as known as instantiation. For example, the object can be created by directly POSTing an HTTP message to the entity's endpoint. For example, the creation of a Thing object can occur via the …/Things path by posting the object's content as JSON encoded message. However, it would also be possible to create a thing as part of creating a datastream. In such a case, the thing is encoded as a JSON object inline to the JSON encoded datastream.

```
{
  "name":"Andreas's Bluetooth Beacon",
  "description": "beacon that measures air temperature and humidity"
}
```

Example 34: Create thing directly via /Things path

```
{
    "unitOfMeasurement": {
        "name": "Humidity",
        "symbol": "RH",
        "definition": "https://byjus.com/physics/unit-of-humidity/"
    },
    "name": "humidity temperature datastream",
    "description": "this datastream exposes the beacon measurements of air humidity",
    "observationType": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM Measurement",
    "ObservedProperty": {
        "name": "Relative Air Humidity",
        "definition": "https://en.wikipedia.org/wiki/Humidity",
        "description": "Air Humidity"
    },
    "Sensor": {
        "name": "urSense",
        "description": "Environment Sensor Board urSense 1.128",
        "encodingType": "application/pdf",
        "metadata": "https://gitlab.DynAikon.com/DynAikontrap/urSense/-/blob/master/ursense-
user-manual-v1.pdf"
    },
    "Party": {"@iot.id": "fe29b5f2-a349-11eb-a344-df45f73dace1"},
    "License": {"@iot.id": "ef3488de-a357-11eb-a344-574838ce4c51"},
    "Thing": {
        "name": "Andreas's Bluetooth Beacon",
        "description": "beacon that measures air temperature and humidity"
    }
}
```

Example 35: Create thing indirectly via /Datastreams path

When implementing access control for the create operation, it must be considered that the object creation can take place one-by-one or as a bulk.

6.3.2 Update Operation

The update operation can only be executed on existing objects using the HTTP PATCH method. The message body contains a partial or full representation of the object using JSON encoding.

Unlike the create operation, the update operation can only be applied to one single object. It is important to reflect the implications to such an HTTP UPDATE request when applied in the context of impersonation. The update of “simple” entity properties might be perfectly OK to allow a user to correct “simple” mistakes.

The use of the update operation and the access conditions may vary from use case to use case and must be wisely authorized on a STAplus endpoint.

6.3.3 Delete Operation

The delete operation can be executed on existing objects using the HTTP DELETE method. According to the STA it is possible to execute a bulk delete or delete one-by-one. When using the bulk-delete option, a database roll-back should be possible in case that a single delete transaction fails. As performance does not matter much for the delete operation, it is recommended to execute single delete operation if possible.

The use of the delete operation and the access conditions may vary from use case to use case and must be wisely authorized on a STAplus endpoint.

6.4 Considerations on Class Party

From the STAplus data model perspective, the use of the Party class is optional. But if leveraged, e.g. to enable a particular business logic like ownership, user authentication should be enforced for create, update and delete operations (HTTP POST, PATCH, DELETE methods). Also, the user should be identifiable by a unique identifier. This user identifier should be stored in the authId property. The enforcement of the create, update and delete conditions on entities linked to the party relies on the uniqueness of the authId.

6.4.1 Single Party Instance

The STAplus data model does not enforce the use of a single Party entity. However, a STAplus implementation should use single instance for the party to represent one user, meaning a 1:1 relationship between unique user identifier and Party entity. This ensures a performant lookup of all observations owned by the same user and prevents database convolution by creating unnecessary digital twins.

6.4.2 Direct Impersonation

In a multi-user deployment leveraging the ownership concept, each user is identified by a user unique identifier (UUID). Impersonation could take place if creation of party with ‘authId’ values was possible where the value is different from the user’s identifier. Also, if the ‘authId’ property of an existing party could be modified, impersonation can take place.

To prevent impersonation, a STAplus implementation should make the ‘authId’ property read-only. Any request via the API to create a party including the ‘authId’ property should result in an HTTP 400 response unless the ‘authId’ is equal to the user’s unique identifier.

Any API request for an authenticated user to update the ‘authId’ should result in a HTTP 403 response.

As a consequence, a STAplus implementation’s business logic is responsible for setting the value of the ‘authId’ property. This could be done automatically by using a unique identifier from the user info provided by the authentication method used (e.g. OpenID, e-mail, etc.).

6.4.3 Indirect Impersonation

When leveraging the STAplus Party class, impersonation of (Multi)Datastream or Group instances is also possible. By modifying the party of a (Multi)Datastream object results in a change of ownership to all observation that are linked to the (Multi)Datastream object. Modifying the party on a group also results in change of ownership but isolated to the group itself.

6.4.4 Delete Operation

The STAplus implementation should consider wisely if it allows to execute a delete operation (HTTP DELETE method) on any Party instance and completely remove the entry from the database. However, to honor the GDPR requirement that a user can “delete account”, an implementation should at least delete the properties name and display name as these are personal information even if the entry cannot or should not be deleted from the database. This would result in a party where the only remaining property is authId. Assuming that the authId cannot be used to resolve personal information, it is acceptable to keep this property.

6.5 Considerations on Class Datastream / MultiDatastream

The integrity of the Datastream as well as the MultiDatastream instances is essential for any productive use. However, some cases have to be considered to ensure integrity when allowing multi-user interactions realizing the ownership concept. Extra care must be taken when implementing the update operation as it affects the integrity of all data objects, linked to (Multi)Datastream instances.

- Class Sensor: The modification of the datastreams or multi-datastreams property would have the same effect as directly modifying the party property of the (Multi)Datastream instance.
- Class Thing: Identical to the Sensor class, the modification of the datastreams or multiDatastreams property would associate the thing and the linked locations with a different datastream.

The (Multi)Datastream class has a property party. To prevent impersonation, this property shall be read-only for API requests.

Any API request to create a (Multi)Datastream or Group instance can either link to an existing party or include a JSON representation for the party in the body of the HTTP POST request. Any request that links to an existing party is good practice. However, it requires some “intelligence” to the calling application as it needs to be able to identify the party beforehand that represents the acting user. It is good practice for a STAplus implementation to reuse existing Party instances to eliminate redundancy. The mapping can take place by unique user identifiers. In case of the first initial creation request, where there is no party representing the user, the STAplus implementation must, of course, create one. But any sequential requests should be linked to the existing party.

6.6 Considerations on Class Thing

According to the SensorThings data model, it is possible to create a thing without being linked to a (Multi)Datastream instance. Also, it is not required that the thing has a location (which links to a location). When operating SensorThings API in read-only-mode, this is not a problem as the thing’s locations will only be set by the authority. The dominant example is a satellite, for which a thing gets created once and where its location is periodically updated by some background process.

For STAplus, leveraging the ownership concept simplifies the use of thing and location. In particular, it is not possible to share the same thing among different users as it would be unclear which user has the permissions to set the thing’s location. The example of the Camera Trap illustrates the problem.

In order to stay compliant with the original SensorThings data model but still support the Camera Trap use case for example, a thing must know which user has the permission to set the location. The user with the privilege to set the location must not necessarily be the same user who created the thing. However, to realize the Camera Trap use case, it is required that each user “owns” a digital twin of the thing. That user is then entitled to set the location of the thing.

As a consequence, a STAplus implementation should prevent the existence of isolated thing. Preventing the existence of isolated thing requires that a create operation on a thing is denied if no (Multi)Datastream instance is included in the request that is linked to a party. As an alternative, the implementation could assign the user's identifier with the thing when it gets created. Any request to set the thing's location afterwards can then be restricted to the owning user.

It is recommended that a STAplus implementation returns an HTTP status 400 for requests that would create thing(s) in isolation.

Connecting a (Multi)Datastream instance to a thing requires to set the thing property. Assuming that a thing has an owner that can update the location (see section 6.6.1), it is possible to attach "my" datastream to some other user's thing. The owner of the thing determines if it is allowed to connect a datastream of another party.

6.6.1 Update

A thing is associated with a user. This user should be able to make changes, to be able to correct typos etc., until there are no datastream(s) connected. As soon as the first datastream is connected, the user can only update the location property. Streamlined with the thinking from above, the set location operation can only be executed by the user that is associated with the thing.

To ensure integrity, it should not be possible to update the datastreams property.

6.6.2 Delete

If a STAplus deployment should allow the delete operation on thing(s) depends on many factors, coined in the business logic.

6.7 Considerations on Class Sensor

According to the SensorThings data model, the Sensor class can be instantiated in isolation, which means that it is not required to be linked to a datastream. Unlike the Thing class, the Sensor class has no property that has the potential to get updated over time.

The realization of the ownership concept does not require that the Sensor instances are associated with a user and should be re-used. To ensure integrity as the basis for re-use, it should be possible to update a sensor only until linked to the first datastream. Delete operation should be disabled and an HTTP DELETE request should result in an HTTP 405.

6.8 Considerations on Class FeatureOfInterest

Instances of the SensorThings data model class FeatureOfInterest can be created in isolation. They can (should) act as global or system wide objects even though instances get created by requests via the API. Therefore, the use or re-use of FeatureOfInterest instances depends on the logic of the calling application. However, from a server-side implementation and the discussion with STAplus, two extremes should be considered:

- There is only one single instance of a FeatureOfInterest that gets created by a particular application. All users of the application generate observations that share the

same feature of interest, because the application logic has a particular logic to re-use the same object.
- Each observation creates a new FeatureOfInterest instance.

It is good practice to re-use these objects as much as possible. The Camera Trap example re-uses the observed area (e.g. a garden) as long as the device is not re-located to another garden.

Any user or application that considers a FeatureOfInterest to be adequate for their purpose should link to it. To ensure integrity and reliability, a STAplus implementation should disable the delete operations and allow update operations on FeatureOfInterest instances until the first observation is linked.

6.9 Considerations on Class ObservedProperty

Instances of the class ObservedProperty can be created in isolation and be understood as common or global wide objects for which re-use is key when trying to find or compare datastreams. Like the FeatureOfInterest class, instances can be created by any user for the common good. To ensure integrity and reliability, a STAplus implementation should enforce the same access for update and delete operations as recommended for FeatureOfInterest: disable the delete operations and allow update operations on ObservedProperty instances until the first datastream/multi-datastream is linked.

6.10 Considerations on Class License

The STAplus extension defines the class License to facilitate better re-use of existing observations. When creating a datastream, the user can associate a license. This license then determines the license for each linked entity and in particular observations that the datastream produces.

To prevent the creation of incompatible and a convolution of licenses in a deployment, a STAplus implementation should generate a pre-defined set of available licenses when being installed. For example, an operator of the STAplus may decide to only allow Creative Commons based licenses or provide configuration options that enable different sets of pre-defined compatible licenses.

The advantage of considering licenses as globally defined read-only objects is that filtering by license hierarchy and compatibility is possible. An implementation should provide pre-defined sets of licenses that can be compared towards equality, containment and incompatibility. For example, “licenseTypes=CC” and “licenseVersion=4” would create all license instances of CC for version 4. The implementation must make sure to be able to determine the equality, containment and incompatibility between licenses. This aspect is important as discussed in the following section “Considerations on class Group”.

6.11 Considerations on Class Group

The STAplus data model defines the Group class as a container for any observations. According to the STAplus data model, a group can be created in isolation. A group may be created in combination of an observation event (for example to bind multiple parts of an observation), or sometime later, for example to package observations for whatever reason.

When realizing the ownership concept, the group can be associated with the acting user, but that is optional. If a party is linked, this user then becomes the de-facto caretaker of the group that is exposed via the API.

The runtime property of the group should be used to declare a group as “closed”. This implies that other users (those that do not own the group) can no longer add observations or update or delete already linked observations. Each time the group owner updates the group or adds an observation, the <endTime> of the runtime property should be updated.

When creating a `Group` object for the purpose of re-use, it is important to differentiate that Observation entities can either be included “inline” or “linked.

```
{
    "name": "Camera Trap Event",
    "description": "Camera Trap Event #1",
    "creationTime": "2021-04-22T18:10:00Z",
    "runTime": "2021-04-21T12:00:00Z/2021-04-22T15:43:00Z",
    "License": {"@iot.id": "CC BY"},
    "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
    "Observations": [
        {
            "phenomenonTime": "2020-05-26T23:00:00.000Z/2020-05-27T23:00:00.000Z",
            "resultTime": "2021-04-22T15:43:00Z",
            "result": [
                1.3,
                87.5,
                980.02
            ],
            "MultiDatastream": {"@iot.id": 2}
        }
    ]
}
```

Example 36: Example of a POST request that adds an observation “inline”

```
{
    "name": "Camera Trap Event",
    "description": "Camera Trap Event #1",
    "creationTime": "2021-04-22T18:10:00Z",
    "runTime": "2021-04-21T12:00:00Z/2021-04-22T15:43:00Z",
    "License": {"@iot.id": "CC_BY"},
    "Party": {"@iot.id": "ff1045c2-a6de-31ad-8eb2-2be104fe27ea"},
    "Observations": [
        {"@iot.id": 1}
    ]
}
```

Example 37: Example of a POST request that adds an Observation “linked”

The use of the License with a group introduces two possible logical interpretations:

1) The License determines the most restrictive license for any observation of the group
2) The License expresses the re-use conditions for the group itself

Re 1)
A STAplus implementation must ensure that only those observations get inserted into or linked to a group of which the license is compatible with the license of the group itself. The license of the group acts as a condition to which observations can be pushed. In other words, the group license determines the maximum restriction of licensing that applies to all observations. For example, a group with no license may contain any observation, regardless of whether the datastream is linked to a license or not. If for example the license of the group

is CC-BY-ND, then only observations can be pushed with a less restrictive license: CC0, CC-BY and CC-BY-ND. Observations of which the datastream is linked to CC-BY-ND-NC cannot be pushed to the group as that license is more restrictive (not inclusive to CC-BY-ND).

When searching for groups, the linking to a license has the big advantage that filtering by group license simplifies the filtering as the overarching license is already set and there is no need to examine each observation's license individually.

Re 2)
When re-using the relations and observations comprised in a group, the license of the group must be honored. For example, if the group gets used to train an AI network and the group has a non-commercial license, then the trained AI network may not be used commercially.

6.12 Considerations on Groups versus MultiDatastreams

It is common to capture several observations about the same feature-of-interest. Mechanisms to relate these observations are necessary. The STA introduces the concept of MultiDatastream that allows for some variable values being captured together or simultaneously, where each one is form a different ObservedProperty instance. STAplus introduces the concept of grouping that allows for some observations to be related.

Typically, a MultiDatastream represents a set of variables that are captured simultaneously and considered as a single observation. In this case there is an assumption that a single thing run by a single party provides the variable values on a single feature-of-interest. A common use case is a weather station that is providing temperature and humidity at the same time.

In contrast, the Group can be used to associate measurements that have differences beyond the ones allowed by a MultiDatastream. Examples are observations on the same feature-of-interest done with more than one sensor, by one or more parties. For example, we can provide weather measurements and a species identification on the same place given as a feature-of-interest). We can also provide confirmation or contradictory observations about the same observed-property by different parties.

The use of the Group class represents a common practice in Citizen Science where several experts provided species identifications based on a picture of a plant or animal. The number of confirmations of the same identification increases the confidence on the observations. More advanced usages of the Group class can be described in combination with Relation class.

6.13 Considerations on Class Relation

The STAplus class Relation can be created on any existing observation. Relations act as global objects for supporting the common good of searching and feeding into a semantically enriched API. The main purpose for searching is to be able to filter on existing relations which simplify the GET query and potentially improve performance and uniqueness. Strengthening the uniqueness is important as it helps to determine semantics between observations that otherwise would have been difficult to detect.

A relation has no user and cannot be created in isolation. It must be directly attached to two observations or to one observation and an external object. Its inclusion in a group can be done when creating the relation or at any later time.

The ability to link two internal observations with each other or to link an internal observation with an external object, e.g. a SKOS definition, strengthens the STAplus model to ensure semantic interoperability.

6.14 Considerations on Class Observation

With leveraging the STAplus data model and in particular the Party and License classes, the SensorThings data model is enriched by expressing ownership and re-use conditions for observations.

The business model of the operator of a STAplus endpoint determines whether it is allowed to update or even delete observations. The operator has to take care of the implications when allowing update or delete operation.

Implications to update and delete may surface with groups, relations etc. as a delete of an observation participating in a relation may invalidate this relation. Such implications to update and delete should be framed in the operator’s terms of use.

7. SensorThings Convenience API

The use of STAplus with the Camera Trap introduced an API requirement that is not implemented properly with the SensorThings API v1.1: Upload of binary observations.

The SensorThings API v1.1 would allow to upload a binary observation such as an image or video only as Base64 encoded data inside the result property of the JSON encoded data. This is inadequate for large images and in particular videos.

The ObservationUpload (/$observation) API is implemented for the FROST-Server implementation. It accepts an HTTP Multipart request where the first part contains the JSON encoding of the observation, but the result property is empty. The first part is stored temporarily. Next, the second part containing the binary data is processed. The implementation stores the binary data on a given CDN (Content Delivery Network) and inserts the resulting URL as the value of the result property on the temporarily stored observation. Finally, the temporarily stored observation is processed.

This implementation is available from GitHub (currently private repo): https://github.com/securedimensions/FROST-Server-ObservationUpload

52°North has implemented this convenience API under the /$upload path.

```
curl -i -L -X POST \
   -H "Content-Type:multipart/form-data" \
   -F "observation={     \"phenomenonTime\": \"2021-05-28T02:45:00Z\",     \"resultTime\":
\"2022-02-08T08:45:00Z\",     \"result\": \"\",     \"parameters\": {         \"tilt_angle\":
\"30\",         \"distance\": \"5\",         \"shutter\": \"2.4\",         \"speed\":
\"1/400\"     },     \"Datastream\": {\"@iot.id\": \"5\"} }" \
   -F "file=@\"./Tongue.png\";type=image/png;filename=\"Tongue.png\"" \
 'https://cos4cloud.secd.eu/sta4cs/v1.1/$observation'
```

Example 38: Example CURL request to upload a binary observation

8. Implementations

As a proof of implementation, different implementations of STAplus exist. Implementations from Secure Dimensions and 52°North focus on the service side. A visualization client application which demonstrates the visualization in a Web-Browser application was implemented by CREAF and Secure Dimensions. A client application that organizes and uploads Camera Trap data was implemented by Secure Dimensions and 52°North. Finally, a data loader implemented by CREAF allows to organize and store Natusfera data in a STAplus deployment.

In addition, INRIA deployed a STAplus endpoint and a data loader that loads approx. 10 million observations from the Pl@ntNet deployment to demonstrate feasibility and research performance.

8.1 Fraunhofer's FROST-Server extension by Secure Dimensions

The Secure Dimensions implementation of the STAplus data model extension is available from GitHub: https://github.com/securedimensions/FROST-Server-PLUS

That STAplus extension is implemented as a plugin to the Fraunhofer FROST-Server develop-2.0 branch that is also available from GitHub: https://github.com/FraunhoferIOSB/FROST-Server/tree/develop-2.0

This implementation offers to enable different aspects of the business logic as introduced in previous sections of this document:

- Concept of Ownership can be switched on by setting the **plugins.plus.enable.enforceOwnsership=true**
- Enforcement of the Group license to control which observations can be added to the group based on their own license can be enabled **plugins.plus.enable.enforceLicensing=true**

This implementation supports different types for the "@iot.id": LONG, STRING, UUID. The type for the ID can be configured as documented for the FROST-Server. To be able to switch-on the enforcement of ownership, the implementation uses the STRING type for Party.@iot.id.

This implementation uses the STRING type for License-@iot.id and generates the following seven Creative Commons licenses during installation:

- CC_PD
- CC_BY
- CC_BY_NC
- CC_BY_NC_ND
- CC_BY_SA
- CC_NY_NC_SA
- CC_ND

When enabling the 'enforceLicensing' option, it is not possible for users to create new licenses. The license compatibility for the Group.License <-> Observation.(Multi)Datastream.License is based on the Creative Commons licenses compatibility chart, available here.

All entities from common classes can be created by any user and remain read-only afterwards.

Enabling the enforceOwnership behavior triggers the following logic reducing the otherwise unlimited create, update and delete operations:

- Party: linked user can trigger update and delete. The delete causes the properties name and ‘displayName’ to be set to empty string. The party itself will not be deleted. The property ‘authId’ will keep the user’s “remote user” value as set by the authentication plugin via the HTTP principal. The update can be executed on the properties name and ‘displayName’. It is not possible to change the ‘authId’ nor the role.
- Thing: The ‘location’ property can only be updated by the user linked to the thing.

Enabling the ‘enforceIntegrity’ behavior triggers the following logic:

- Location: A location can be created by any user. Any location that is not (yet) linked to a thing can be updated and deleted. Once the location is linked to a thing, it can no longer be updated nor deleted.
- ObservedProperty: Similar to Location. Once linked to a datastream, it can no longer be updated nor deleted.
- FeatureOfInterest: Similar to Location. Once linked to an observation, it can no longer be updated nor deleted.
- Group: If the ‘enforceOwnership’ option is enabled, the creation of a group requires to set a party. Otherwise, the group can be created by any user. Once adopted by a user (associated with a party entity), the following CRUD conditions are enforced: <endTime> for the runtime property can only be updated by the associated party. Observations can be added to a group until the <endTime> for the runtime property is set. A license can only be set to the group by the linked user until the first observation or relation is added. Then, the license cannot be changed nor deleted any more.
- Relation: Can be created, updated and deleted by any user until it is linked to a group. Then, only the owner of the group can update or even delete the relation.

This implementation does not prune un-used instances of the Sensor, Location, Project, ObservedProperty, FeatureOfInterest classes.

8.2 Fraunhofer’s FROST-Server convenience API by Secure Dimensions

The SensorThings API supports the creation of entities via JSON encoded requests. This format is not suitable for binary observations such as images or (short) videos.

For the Camera Trap use case, an animal detection (camera trap) event consists of at least one image or video that captures the detected animal. To support easy management of the entire data generated – video / image, taxon information, user’s comment, environmental sensor measurements – Secure Dimensions has implemented yet another plugin for the Fraunhofer FROST-Server base.

The implementation supports face detection via OpenCV as an experimental feature.

This STA convenience API is an independent contribution from the STAplus work; the ObservationUpload API does **not** require STAplus data model to be present. The

implementation is available from GitHub: https://github.com/securedimensions/FROST-Server-ObservationUpload.

8.3 STAplus BASH client for Camera Trap data upload by Secure Dimensions

The application that uploads data from the camera trap to STAplus is implemented as a BASH script. The implementation is available as open source: https://github.com/securedimensions/DynAikonTrap/tree/master/STAplus

8.4 52°North

The 52°North implementation of the STAplus data model extension is available from GitHub: https://github.com/52North/sensorweb-server-sta

8.5 STAplus JavaScript Web-App by CREAF

The code for the JavaScript app that transforms the Natusfera/iNaturalist observations into STAplus observations is available via GitHub as open source: https://github.com/joanma747/Natusfera4STAplus.

The code for the MiraMon Map Browser with support for STAplus to request and present the results as data points visible in the screen in combination to other data is available via GitHub as open source:
https://github.com/grumets/MiraMonMapBrowser

In both cases, in order to support the execution in a web browser, the STAplus service should support CORS interactions.

8.6 Pl@ntNet as a Service by INRIA

The implementation of Pl@ntNet data as a Service is self-hosted. It relies on FROST-Server by Fraunhofer Institute, and FROST-Server-PLUS plugin by Secure Dimensions.

The Pl@ntNet STAplus endpoint is available and documented here: https://my-api.plantnet.org/#/SensorThings/getV2StaplusPath
As this service is currently protected by an authentication mechanism, a demonstration endpoint with a subset of data is publicly available at https://thymerais.cirad.fr/cos4cloud/api-demo/v1.1

Table 4 presents a few of the most common queries in a citizen observatory use case. They were used as a test set for performance measurement, and raised performance limitation issues with the default PostgreSQL backend configuration. Those limitations were addressed by adding PostgreSQL indexes, as stated in the following section 8.7.

Observations from species 'Acer campestre L.', top 100	/Groups?$filter=(Observations/Datastream/ObservedProperty/name eq 'Taxon' and Observations/result eq 'Acer campestre L.')&$orderBy=created desc&$expand=Observations/Datastream/Party
Observations having a suggested determination, top 10	/Groups?$filter=(Observations/Datastream/ObservedProperty/name eq 'Taxon' and length(Observations/result) gt 0)&$orderBy=created desc&$top=10&$expand=Observations/Datastream/Party
Undetermined observations, top 10	/Groups?$filter=(Observations/Datastream/ObservedProperty/name eq 'Taxon' and length(Observations/result) eq 0)&$top=10&$expand=Observations/Datastream/Party&$orderBy=created desc
Observations from PN user 102555894, top 100	/Groups?$expand=ObservationRelations,Observations/FeatureOfInterest,Observations/Datastream/Party,Observations/Datastream/License,Observations/Datastream/Project&$filter=Observations/Datastream/Party/authId eq 102555894
Observations in 'Montpellier', top 10	/Groups?$filter=(Observations/FeatureOfInterest/properties/locality eq 'Montpellier')&$orderBy=created desc&$top=10&$expand=Observations/Datastream/Party,Observations/FeatureOfInterest
Observations at less than 1km from a given place, top 10	/Groups?$filter=geo.distance(Observations/FeatureOfInterest/feature, geography'POINT (3.88 43.608)') lt 0.01&$orderBy=created desc&$top=10&$expand=Observations/Datastream/Party,Observations/FeatureOfInterest
Observations in a polygon matching Montpellier city (st_within), top 10	/Groups?$filter=st_within(Observations/FeatureOfInterest/feature, geography'POLYGON ((3.88 43.51, 3.78 43.6, 3.86 43.7, 3.955 43.638, 3.965 43.55, 3.88 43.51))')&$orderBy=created desc&$top=10&$expand=Observations/Datastream/Party,Observations/FeatureOfInterest
Observations sorted by date decreasing, top 100	/Groups?$orderBy=created desc&$expand=Observations/Datastream/Party,Observations/FeatureOfInterest&$top=100
Observations on may 20 2018, top 10	/Groups?$filter=Observations/phenomenonTime gt 2018-05-20T00:00:00.000Z and Observations/phenomenonTime lt 2018-05-21T00:00:00.000Z&$orderBy=created desc&$top=10&$expand=Observations/Datastream/Party,Observations/FeatureOfInterest

Table 4: Common STAplus queries

8.7 PostgreSQL backend indexation

Using the data model mapping described in the Best Practice section 5.5, 11,660,377 PN observations are currently stored as 165,271,049 PostgreSQL tuples (including 39,508,041 Observations and 27,807,745 Relations), for an average of around 14 tuples per PN observation. The dataset volume on disk is 47 GB.

Using PostgreSQL as a backend, it happened that numerous SensorThings API requests corresponding to classic use-cases (see Table 4) lead to slow SQL queries. Time-measured unit test results conducted us to add a few indexes to different PostgreSQL tables, which reduced execution time by a factor of up to 1000, depending on the query.

Note: FROST-Server is set to use Long IDs in all tables (not UUIDs).

a) `CREATE INDEX "OBSERVATIONS_RESULT_STRING" ON "OBSERVATIONS" using btree ("RESULT_STRING");`
b) `CREATE INDEX "OBSERVATIONS_PHENOMENON_TIME_START" ON "OBSERVATIONS" using btree ("PHENOMENON_TIME_START");`
c) `CREATE INDEX "OBSERVATIONS_PHENOMENON_TIME_END" ON "OBSERVATIONS" using btree ("PHENOMENON_TIME_END");`
d) `CREATE INDEX "DATASTREAMS_PARTY_ID" ON "DATASTREAMS" using btree ("PARTY_ID");`

```
e) CREATE INDEX "PARTIES_AUTHID" ON "PARTIES" using btree ("AUTHID");
f) CREATE INDEX "FEATURES_GEOM" ON "FEATURES" using gist ("GEOM");
g) CREATE INDEX "FEATURES_PROPERTIES_LOCALITY" ON "FEATURES" using btree (("PROPERTIES"#>>'{locality}'));
h) CREATE INDEX "GROUPS_CREATED_DESC_ID" ON "GROUPS" using btree ("CREATED" desc, "ID" asc);
i) CREATE INDEX "GROUPS_NAME_DESC_ID" ON "GROUPS" using btree("NAME" DESC, "ID");
```

Example 39: SQL queries to accelerate performance of the implementation by adding indexing to the database

8.8 SensorThings API Map Library

The JavaScript Library STAM (SensorThings API Map), available from https://github.com/DataCoveEU/STAM, allows to visualize sensor data provided via a SensorThings API v1.1 service endpoint. Because the STAplus Data Model is a natural extension to the SensorThings Data Model, it is possible to use this library as-is!

The use of this library eases the creation of a Web-Mapping application leveraging the popular JavaScript based mapping frameworks Leaflet or OpenLayers. A demonstration prototype is illustrated in section 5.6.2.

The STAM library can also be used to connect to access protected STA and STAplus endpoints using an HTTP Header or URL parameter-based approach. The following STAplus Viewer App (https://cos4cloud.secd.eu/staplus-viewer-appx) demonstrates how to connect to a STAplus endpoint that is protected via an OAuth2 Access Token and the use of a proprietary URL parameter.

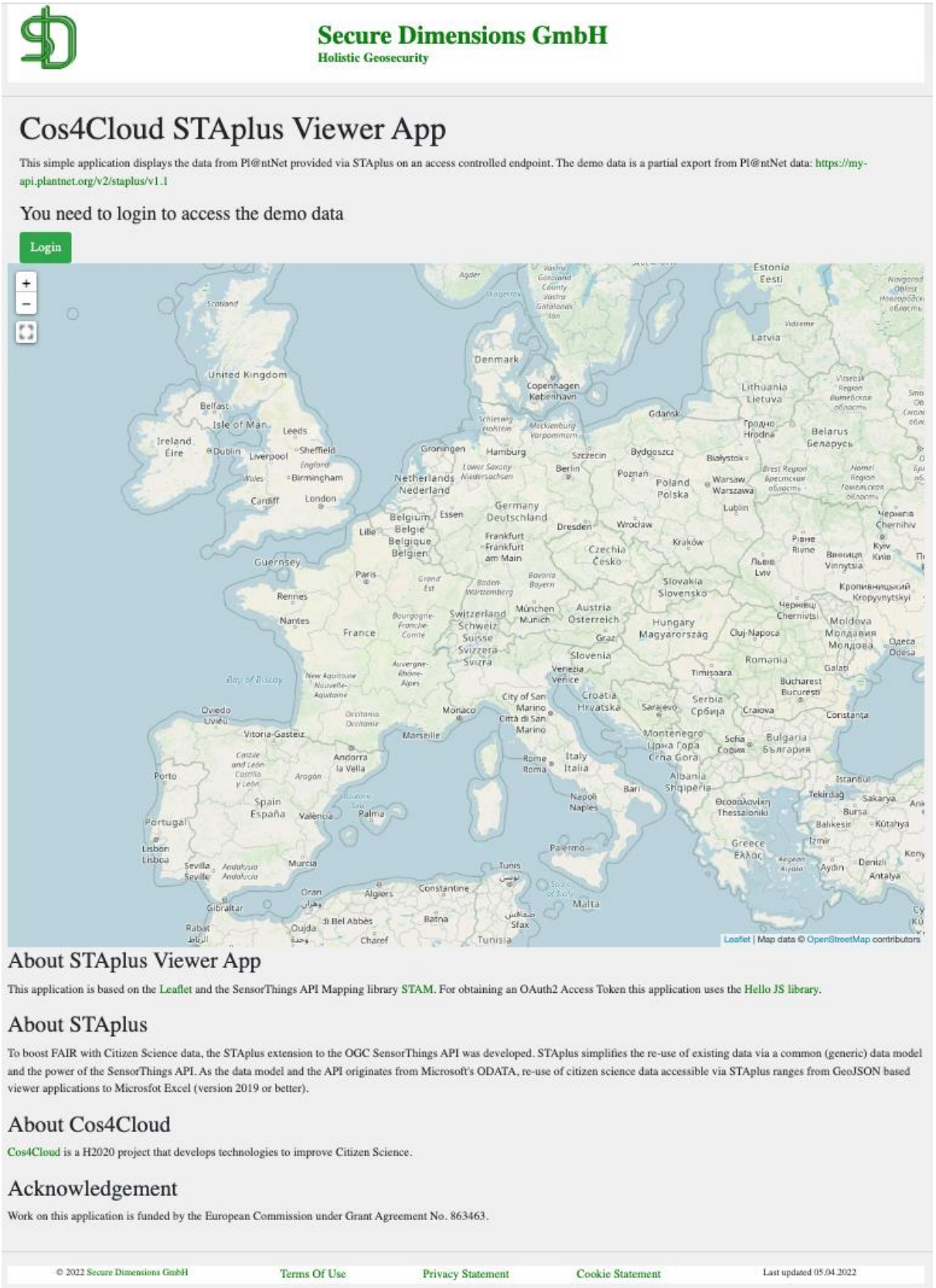

Figure 23: STAplus Viewer App connecting to an access protected STAplus endpoint

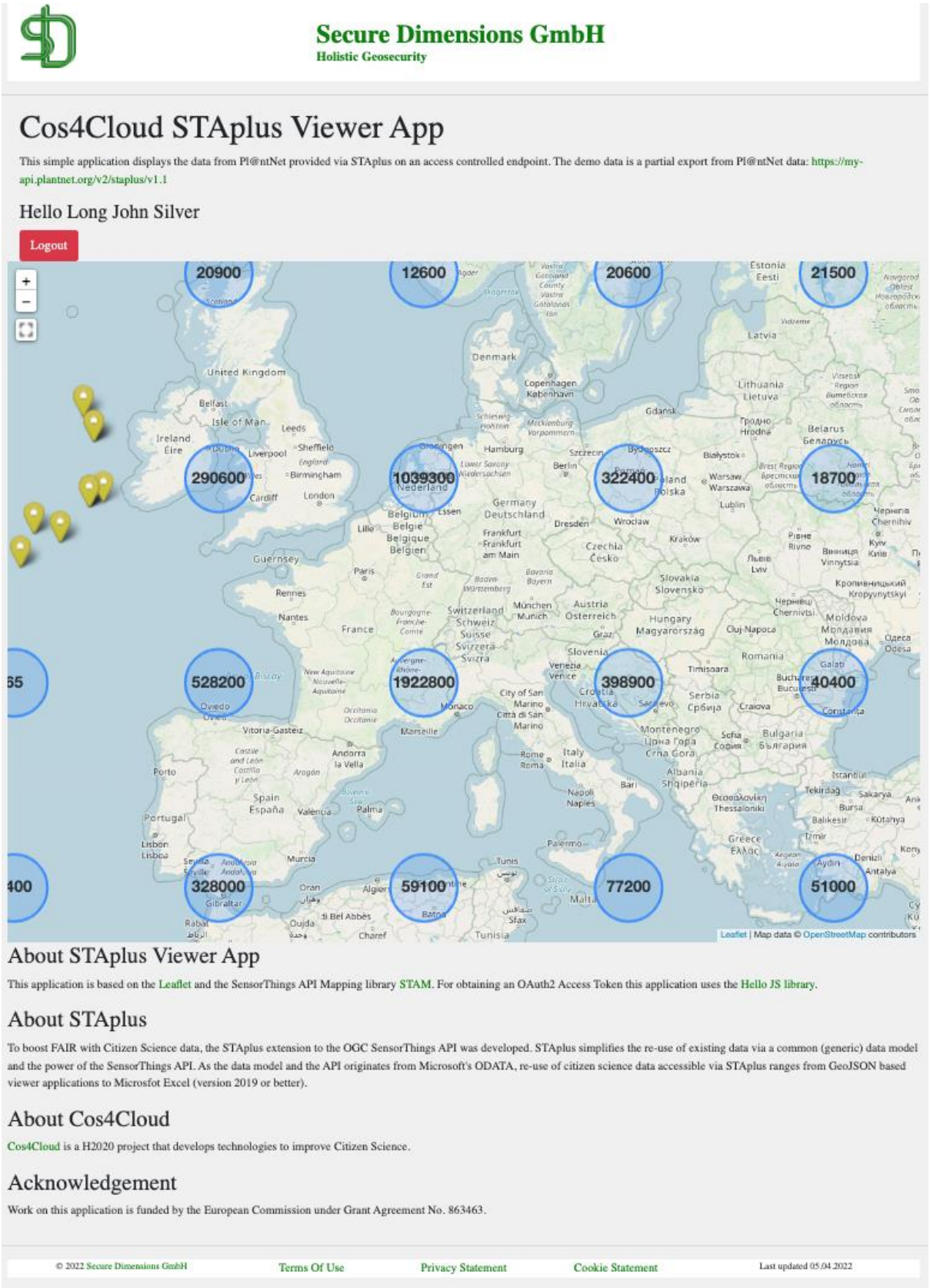

Figure 24: STAplus Viewer App connecting to an access protected STAplus endpoint, displaying FeaturesOfInterest

8.9 Excel OData Data Feed

Microsoft Office Excel version 2019 supports to load data via an OData source. Figure 25 illustrates the similar information as visualized by the STAplus Viewer App for a FeatureOfInterest (ID 9357).

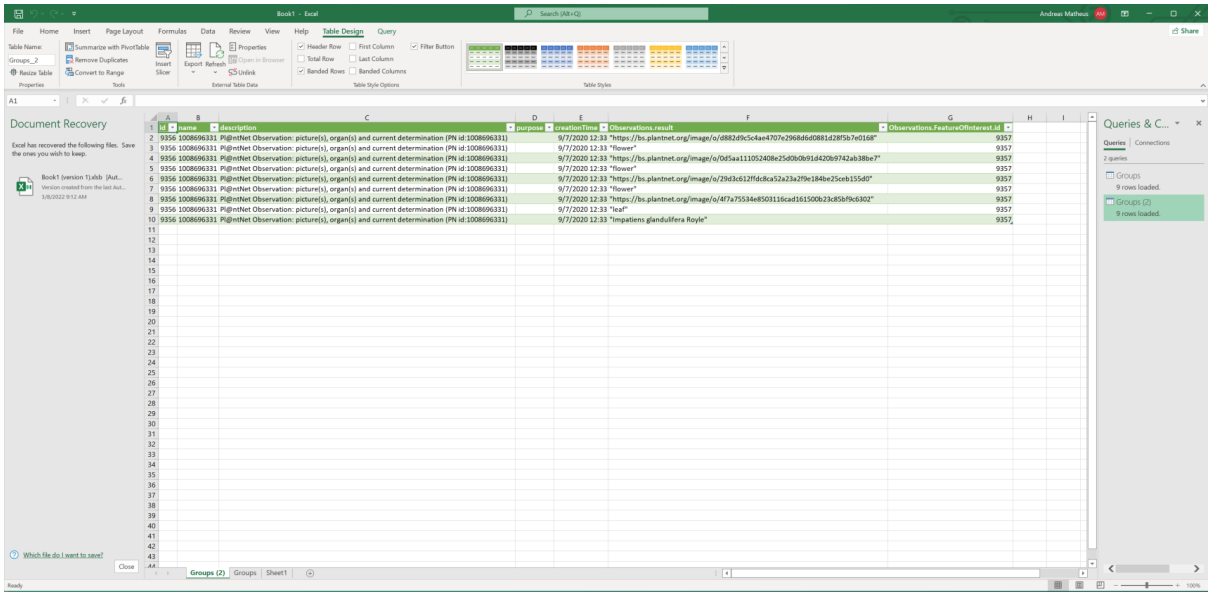

Figure 25: Microsoft Office Excel version 2019 view of the STAplus data

Note: The use of Microsoft Excel is a time-consuming effort as the "Power Builder" seems to load ALL data from the Groups multiple times: (i) when the Power Builder is started, (ii) when the filter constraint is adjusted, and (iii) when the "load data" button is finally clicked. To produce the excel sheet above (see Figure 25), it took approximately 5 minutes on a VM with 2vCPUs and 4GB of memory.

The same information is obtained by the STAplus Viewer App upon a click on a marker within a fraction of a second. The main reason is that the Excel Power Builder loads the entire data to allow the user to filter with live values.

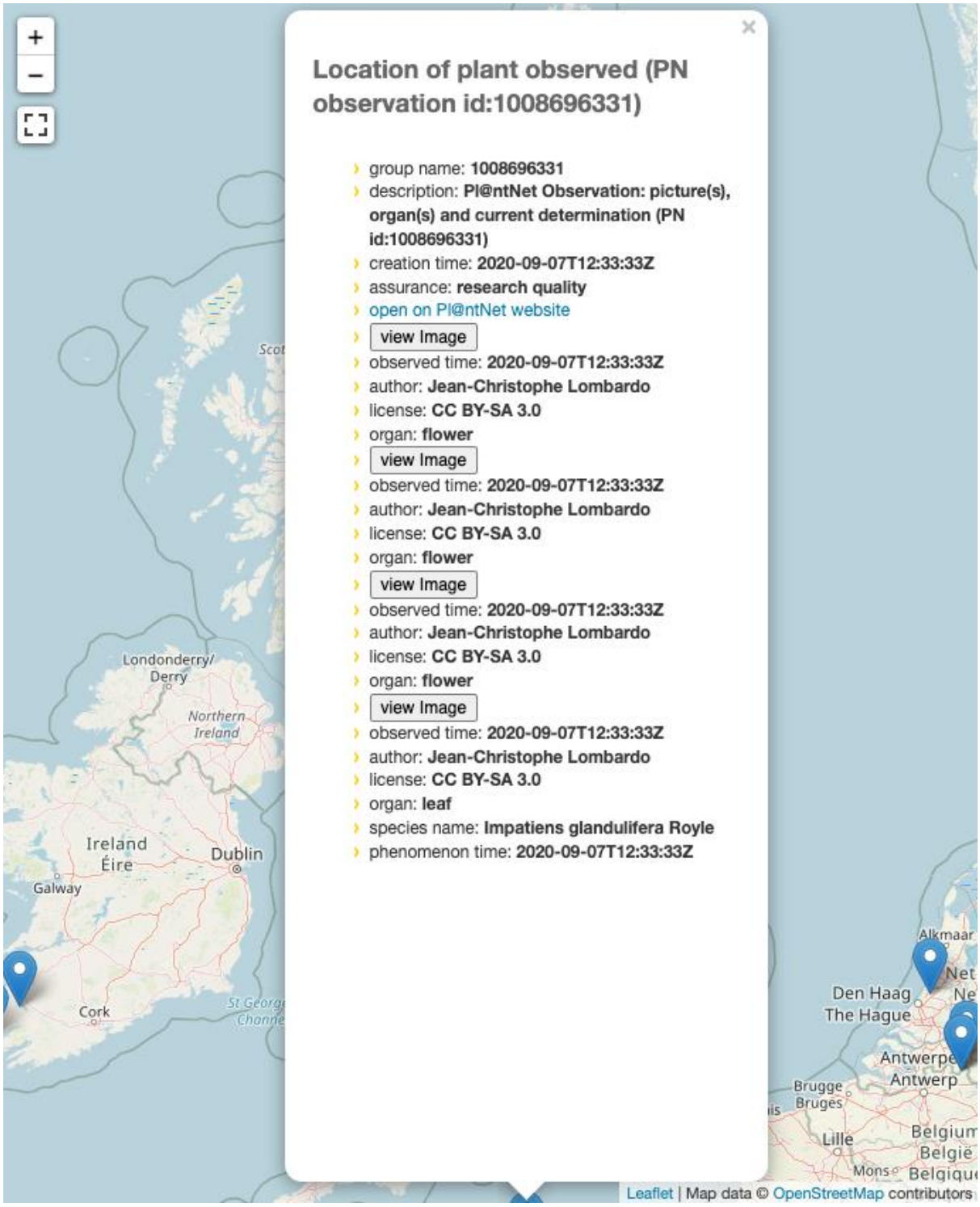

Figure 26: Visualization of the Group observations for FeatureOfInterest (ID 9357)

9. Future Work

9.1 Technical Aspects

Future work on STAplus can be separated into work on the generic Data Model, the API or on the Business Logic.

To increase technical and semantic interoperability, work on further enhancing the Data Model may be stimulated by other use cases from Citizen Science such as

- Biodiversity domain – from existing data models described in the BP use cases (e.g. Natusfera, Pl@ntNet, other GBIF-based platforms)
- Environment domain – from promoting new BP use cases, especially from Cos4Cloud (Cos4Env, CanAirIo+MOBIS, MiniSecchi+MOBIS), which may be well fit to STAplus rather than to Darwin Core API.

Semantically enriched API

- How does the use of JSON+LD enrich the semantical understanding and the meaning of both the core SensorThings Data Model as well as additional attributes provided in the class properties?
- Can (Geo)SPARQL be used to ingest and interpret the JSON-LD representation?
- How can various ontologies (Darwin Core, Dublin Core, SOSA/SSN, SKOS and further definitions from the OGC Definition Server) be integrated?

Further Data Model Enhancement

- Allow Groups to contain Groups
- Allow association from Relation to Party
- Foresee standardized dedicated association for the indication of the likelihood of a species would be valuable, currently not available from existing vocabularies such as Darwin Core.

Interoperable Reporting of existing Access Condition

- Authentication: APIKey or Access Token (common protection)
- Authorization: GeoXACML policies (highly business logic specific)

9.2 Procedural Aspects

For the purpose of standardization, the submitters and the author of this Best Practice will contact the OGC SWE.IoT SWG to accept a draft standard that contains the STAplus Data Model. This draft standard will outline one or more Conformance Classes that allow the interoperable use of STAplus.

Appendix A: STAplus Data Model

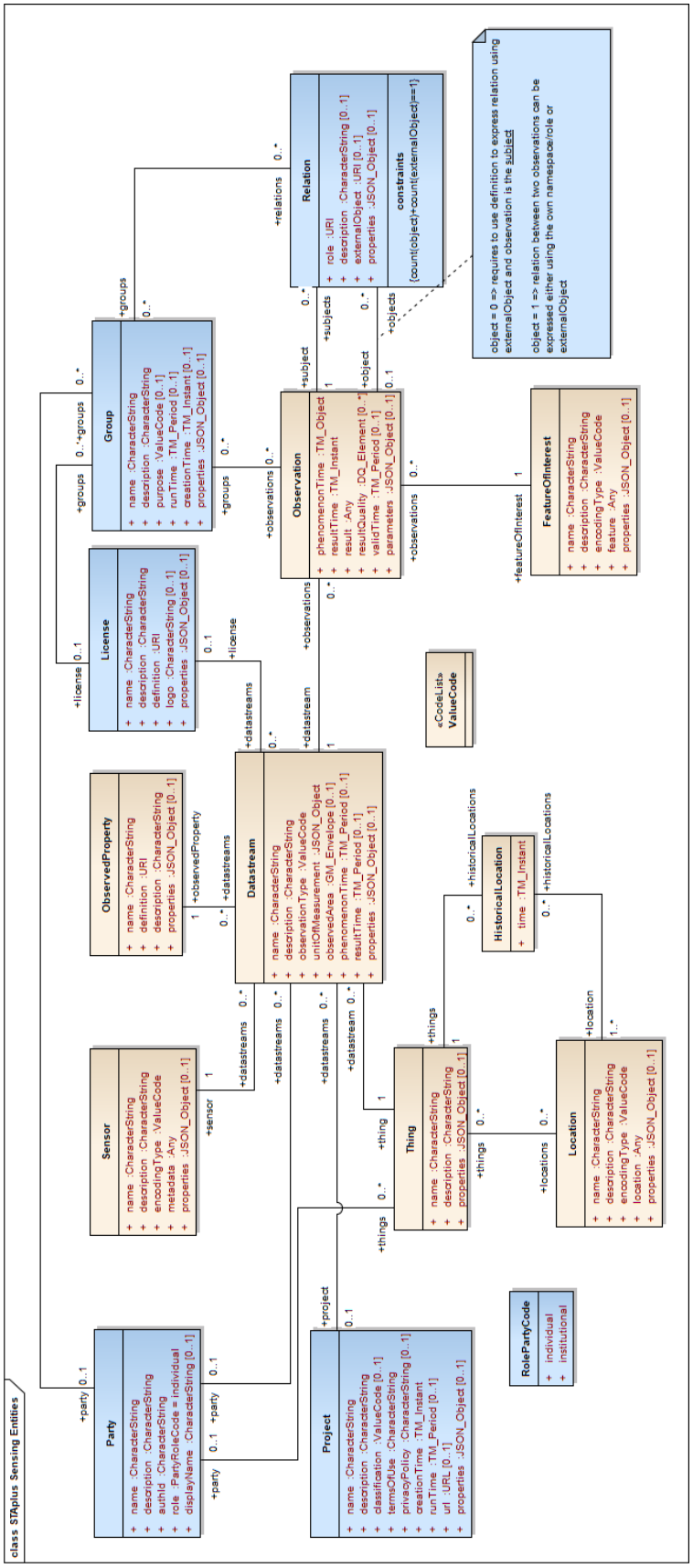

Figure 27: STAplus extension to Datastream

SensorThings data model in yellow – STAplus extension in blue.

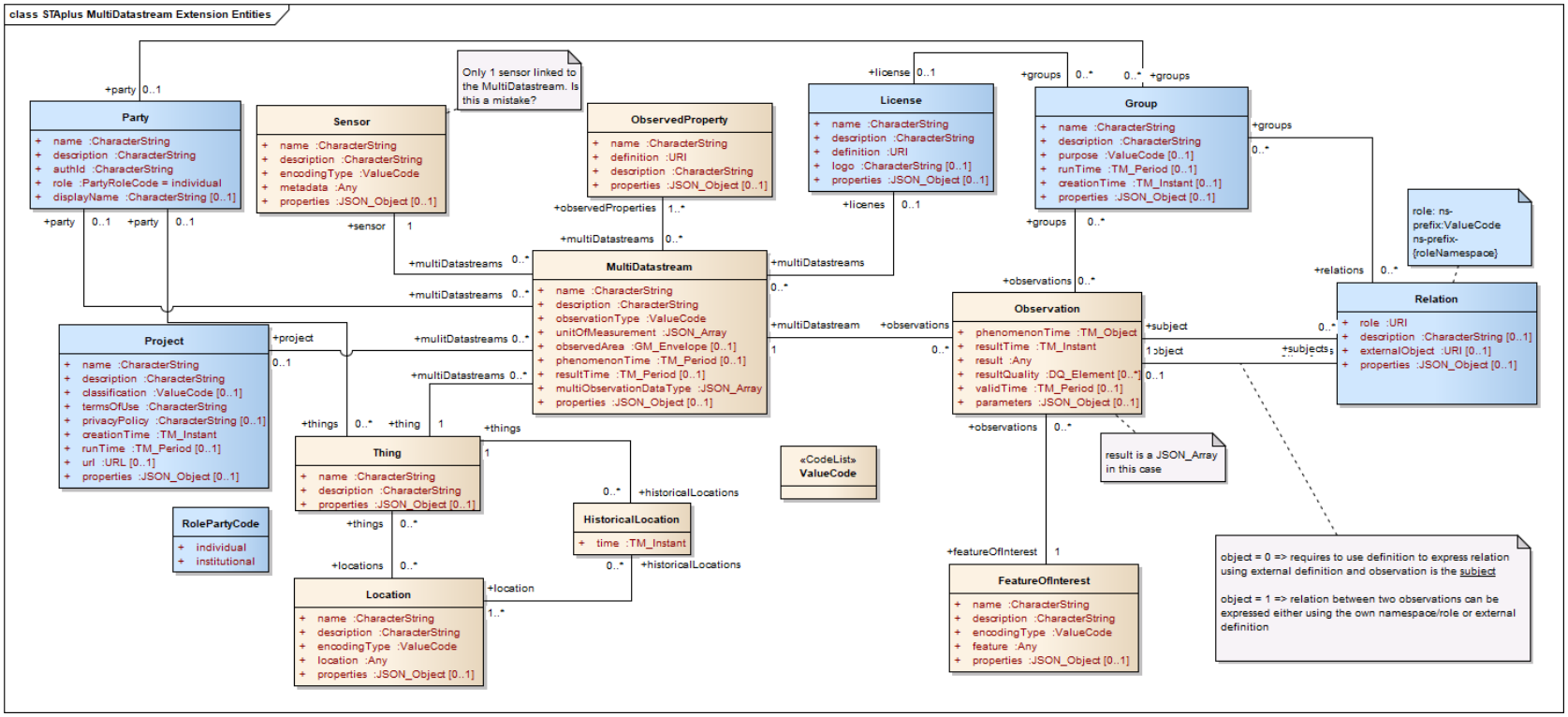

Figure 28: STAplus extension to MultiDatastream

SensorThings data model in yellow – STAplus extension in blue.