A standardised approach for serving environmental monitoring data compliant with OGC APIs

Juan Pablo Duque[1], Angelly de Jesus Pugliese[1], Maria Antonia Brovelli[1]

[1] Department of Civil and Environmental Engineering (DICA), Politecnico di Milano –
(juanpablo.duque,angellyde.pugliese,maria.brovelli)@polimi.it;

Keywords: environmental monitoring,OGC API,open data,geospatial information standards

Abstract

Highlighting the importance of environmental monitoring data and its management, and the usage of standards for providing interoperability to geospatial information, we present and document the implementation of ROSE-API (Reusable Open Source Environmental-Data-Management OGC API), an OGC API-compliant web service for exposing feature data and georeferenced timeseries, as the ones usually generated by environmental monitoring networks. Multiple OGC API standards that align with the goal of exposing environmental data were implemented, in particular OGC API - Features, Environmental Data Retrieval (EDR), and Processes. With interoperability and flexibility in mind, ROSE-API allows discovery and processing capabilities for environmental monitoring data, exposing it in standardised formats and structures suitable for modern internet usage, allowing spatial and non-spatial filtering, and providing server-side processing capabilities. We tested our implementation with 13 years of air quality observations, consisting of more than 50 million data points, to address its compliance with the implemented OGC APIs, performance metrics, and processing capabilities.

1. Introduction

Environmental monitoring is a relevant task in order to cope with climate change. The spatial authoritative data needed for such purpose is generated by different sources, e.g., local or national organisations. In order to assess the environment as a whole it is fundamental to allow interoperability among the different sources of data, hence, standards are required.

The Open Geospatial Consortium (OGC) is an organisation which has been providing geospatial standards throughout the years. OGC standards have been extensively used by local authorities and organisations to share their data. OGC Web Services (OWS) are a family of standards for sharing different formats of spatial data. Although they are used extensively, they are based on technologies that are not suitable for modern internet usage (e.g., XML language); furthermore, OWS services are not indexed by search engines (Santana and Davis, 2023).

The evolution of OWS is a family of standards called OGC APIs. They are based on state-of-the-art web technologies such as REST, JSON, and OpenAPI, which are more suitable for the modern internet. Among the OGC API standards, the OGC API - Features, OGC API - Environmental Data Retrieval (EDR), and OGC API - Processes, are especially useful for managing and exposing georeferenced environmental data. These standards will be further described in the following sections.

Although OGC APIs have not reached mainstream adoption, multiple organisations are already following OGC API standards for exposing their environmental data. Such is the case of the Meteorological Service of Canada (MSC) which exposes historical and real-time weather, climate and water datasets through the GeoMet API (Environment and Climate Change Canada (ECCC), 2020). In addition to these services, testbeds have been implemented for supporting the integration of OGC standards with existing authoritative data, e.g., an OGC testbed platform for Switzerland (Blanc et al., 2022), and the Geonovum OGC API Testbed (Geonovum, 2021).

It is important to acknowledge the complexity of maintaining a service for exposing environmental data, e.g., meteorological data at the regional or national level, and the effort of adopting new standards to comply with the OGC API specifications. Therefore, we propose ROSE-API (Reusable Open Source Environmental Data Management OGC API) as a general framework which follows multiple OGC API standards for data discovery, querying, and processing of common tasks such as outlier removal and data aggregation. This approach can be (re) used by an organisation for ingesting and later exposing their data through OGC API standards.

As a case study, we showcase the use of ROSE-API with the data of ARPA Lombardia (Regional Agency for Environmental Protection (ARPA Lombardia), 2024), the local environmental agency of the Lombardy region in Italy. Air quality monitoring data was ingested and exposed following OGC API standards.

The upcoming sections are three. Section 2 includes the theoretical framework where we define what is the OGC, the standards that we implemented on ROSE-API, and explain how these standards are relevant for environmental data monitoring. Section 3 provides a complete description of the developed software, including the system architecture and components. Finally, section 4 documents the implementation of the aforementioned case study as a test of the capabilities of ROSE-API with respect to the existing ARPA Lombardia API (Open Data Lombardia, 2023).

2. Theoretical Framework

2.1 OGC, OGC Web Services and OGC APIs

The Open Geospatial Consortium (OGC) is an international organisation committed to improve the access to geospatial data. For this purpose, it has proposed over the years a series of standards to apply FAIR principles (Findable, Accessible, Interoperable, and Reusable) to spatial information. Such standards

allow interoperability across organisations and maximise the value of geospatial data. OGC Standards have been used extensively in academia, governmental institutions, and industry. The most established standards are the web services, while the API-based ones are still gaining traction.

2.1.1 OGC Web Services: The OGC Web Services (OWS) refer to a set of standards which define a core model for exposing geospatial data and its processing. The OWS include the commonly used Web Map Service (WMS), Web Feature Service (WFS), Web Coverage Service (WCS), Web Processing Service (WPS), Sensor Observation Service (SOS), among others.

WFS is a standard which provides an interface for requesting georeferenced features. It allows a series of operations for the discovery, querying, management, locking, and transactional operations over feature data. WPS is a standard for defining the requests and responses of geospatial data processing services. It provides processing capabilities to GIS servers by enabling interfaces that execute server-side functions. SOS is a standardized service for the management of sensor networks and access to sensor observations.

This brief definition of the WFS, WPS, and SOS standards was provided to contextualise the reader with those standards and to show in the next section how they are enhanced and technologically updated in the context of OGC APIs.

2.1.2 The OGC APIs – Features, EDR, and Processes: OGC API - Features (Open Geospatial Consortium, 2022) is a multi-part standard which enables the creation, modification, and query of spatial feature data. It provides access to collections of geospatial data. The standard requires a basic set of interfaces or endpoints for exposing the data. The first is `/collections` which lists the available collections that can be eventually queried, alongside their metadata including the id, description, and spatial and temporal extent. A collection can be queried through `/collections/collectionId/items?queryParams`. The request can receive multiple filters through the query parameters, allows pagination, and format selection (e.g., HTML, GeoJSON (feature collection), or GML). A single feature from a collection can be retrieved through the endpoint `/collections/collectionId/items/featureId`.

OGC API - Environmental Data Retrieval (EDR) (Open Geospatial Consortium, 2023b) is an extension of the previously described API that includes multiple ways of spatially querying environmental data. The OGC API - EDR standard includes all the endpoints of OGC API - Features but adds more ways to query data by enabling endpoints such as `/collections/collectionId/position`, `/collections/collectionId/cube`, or `/collections/collectionId/area`.

OGC API - Processes (Open Geospatial Consortium, 2021) is a standard that allows the execution of server-side computational procedures over the collections' data or user input. The procedures ingest spatial data, vector and/or coverage, and return an output according to the specified functionality. The list of processes available can be obtained from `/processes`, the description and metadata of each process at `/processes/process-id`, and can be executed through the endpoint `/processes/process-id/execution`. A process can be synchronous or asynchronous to handle any type of request. Processes are handled through jobs which describe a process execution. A new job is added to the jobs list when a process is executed. The endpoints for handling the jobs are `/jobs` which returns the list of jobs, `/jobs/job-id` which returns the status of a single job, and `/jobs/job-id/results` which returns the result of a job.

2.2 Environmental data monitoring

Environmental data refers to the state of the environment at a specific time and place. It regards, among others, meteorological, air quality, water, biodiversity, and waste. Environmental data monitoring is a practice useful for assessing, analysing and forecasting the trends of environmental conditions (United Nations Economic Commission for Europe (UNECE), 2003). The monitoring of the environment supports decision-makers and helps in the assessment of policy implementation.

The Infrastructure for Spatial Information in Europe (INSPIRE) aims to create a spatial data infrastructure for the European Union with the purpose of sharing environmental spatial information among European public organisations. One of INSPIRE's principles is the possibility to combine spatial information from different sources and share it with users and applications. INSPIRE proposes a set of guidelines for adding specific metadata to a dataset to be discoverable and findable to INSPIRE services and end users. The INSPIRE geoportal provides an overview and access to datasets that are compliant with INSPIRE metadata standards.

INSPIRE recognizes the importance of environmental monitoring as it considers them "High Value" datasets. This includes environmental monitoring facilities and their underlying observations e.g., air quality, meteorological, and marine monitoring networks. This is an initial step for metadata agreement towards interoperability. Following this idea, having services, such as OGC APIs that can provide interoperable data access to multi-source environmental monitoring data is fundamental.

3. Software Description

In this section, we describe the software architecture, the technologies that were used, and the functionalities of each of the components of our OGC API implementation. ROSE-API (Reusable Open Source Environmental-Data-Management OGC API) is a server-side web application for managing, processing, and exposing environmental monitoring data using multiple OGC API standards (Features, Environmental Data Monitoring - EDR, and Processes). With reusability in mind, our API is highly configurable and dynamic.

ROSE-API features several dynamic components. By dynamic we intend to say that metadata and models are built on runtime based on a specific configuration without doing any modifications to the source code.

The dynamic behaviour of ROSE-API includes three strategies, the first is the dynamic generation of collections based on the database tables which store environmental and feature data, the second is the modular aggregation of new processes where user-defined Python scripts with a specific structure are interpreted by the server, and the third is the dynamic generation of the OpenAPI document where the different endpoint descriptions for collections and processes are automatically created.

This implementation is presented as open-source software and relies on multiple open-source technologies such as Python, PostgreSQL, and PostGIS, as well as on open standards such as the aforementioned OGC APIs and OpenAPI.

3.1 System architecture and technologies

ROSE-API showcases a server-side web application connected to a geospatial database for the management and exposure of environmental data. Figure 1 shows the overall software architecture of the system.

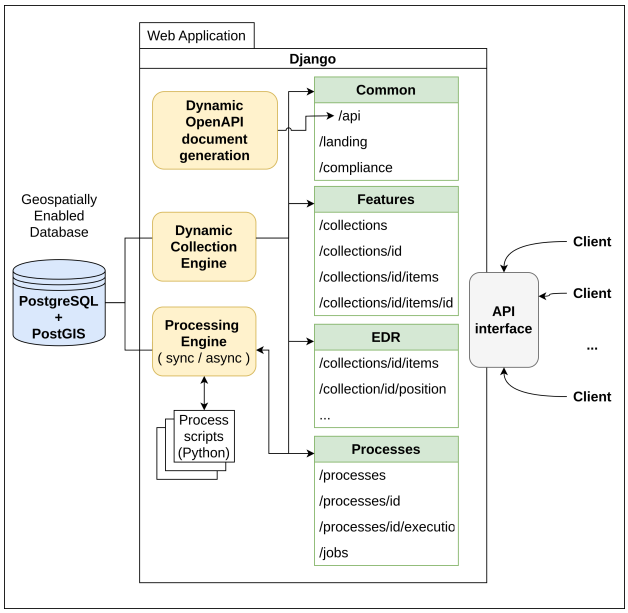

Figure 1. ROSE-API software architecture and components.

The system is composed of two macro components, a geospatial database and a web application. The geospatial database is a PostgreSQL instance where data and configuration parameters are stored. The PostGIS extension is enabled to provide geospatial support, which allows the storage and processing of geometries. The web application provides the API structure and the server functionalities. It was developed with Django, which is a Python-based web framework. It presents a set of dynamic modules alongside a component for each of the OGC APIs that are implemented.

Among the general characteristics of the API, it is worth mentioning the following:

- **Content negotiation**: The format of the response is defined by the client by sending the desired format as an HTTP header or through the "f" query parameter. By default, all endpoints allow responses in HTML and JSON. Any endpoint with spatial capabilities allows GeoJSON responses.
- **Web Linking**: Every response is always accompanied by multiple links (e.g., the same request in a different format, the landing page, the parent request). They work as metadata and allow a smoother navigation through the API for both human users and machines.
- **Pagination**: As some collections may contain massive amounts of data, all endpoints that return lists of elements allow pagination. With pagination, it is possible to retrieve a subset of the whole data that matches a specific set of filters. Pagination is enabled by two parameters, limit and offset. Limit specifies the maximum number of elements to be returned in a response, while offset specifies the position of the first element to be returned, considering that the whole set of elements is a list. In this way, it is possible to retrieve a complete list of elements in "pages". Each page corresponds to a single request. A client is in charge of reconstructing the complete list of elements.

3.2 Components

ROSE-API features the following dynamic modules and components.

3.2.1 Dynamic Collection Engine: Datasets in OGC APIs are based on the concept of collection. A collection is a geospatial resource, such as a feature collection, a set of georeferenced sensor observations, or a GeoTIFF image. Collections describe the data structures that are managed by the server and the individual features or data points that are stored in each of the collections. For ROSE-API, collections are a set of features that could correspond to spatial entities (e.g., a monitoring station, the locations of cities, etc.), but also could correspond to sensor observations or georeferenced timeseries. These collection categories correspond to feature collections and EDR collections, respectively.

For this matter, creating a reusable and configurable OGC API that could potentially manage any kind of dataset structure requires collections to be managed dynamically. The Dynamic Collection Engine allows the definition of a collection using a JSON-based configuration where each field of the collection is described. In this way, collections can be managed independently from the source code. The configuration for each collection is stored in the database and a new table is created to store the features related to the collection. Finally, the Dynamic Collection Engine creates Django models on runtime based on the collection configuration to enable the usage of Django and Python features, such as serializers which format the API responses, and the ORM (Object-Relational Mapper) for querying the database.

3.2.2 OGC API - Common Component: This component exposes some general-purpose endpoints which are specified in the OGC API - Common standard (Open Geospatial Consortium, 2023a). The Common endpoints are one of the main building blocks of the OGC API specification and are shared among all types of APIs. The Common component provides a landing page with information about the server in both human-readable (HTML) and machine-readable (JSON) formats, a list of the conformance classes of the API, and a complete description of the API through a dynamically generated OpenAPI document. The endpoints provided by this component are described in table 1.

3.2.3 OGC API - Features component: This component enables the discovery and management of feature collections. It follows the OGC API - Features specification (Open Geospatial Consortium, 2022) and offers a set of endpoints for the exploration and filtering of feature data. Feature collections can store any kind of vector data, such as points or polygons. Examples of feature collections include lists of sensors or cadastral parcels.

Endpoint	Description
/	The landing page of the API. This is the entry point of the system and provides a machine and human-readable way to explore the API.
/api/	Provides the API description through a dynamically generated OpenAPI document. This document is provided in JSON format following the OpenAPI standard. It is automatically generated using a common base and adding dynamically the collections and processes endpoint descriptions.
/conformance	Provides a list of the conformance classes that this API complies with. Each element of the list points to a URL with the specification of the implemented standard.

Table 1. Endpoint description for OGC API - Common component.

The features component also provides several querying capabilities to extract and retrieve the collections' data from the database. It supports spatial filtering through a bounding box, temporal filtering by date ranges, and attribute filtering by the data fields of the collection. Feature collection data can be retrieved in multiple formats, including HTML, JSON, and GeoJSON. Pagination is also available.

This component not only provides reading capabilities but also the management of collections. It provides creating, retrieving, updating, and deleting (CRUD) capabilities to the collections and the collection's data.

The endpoints provided by the features component are described in table 2, while the filtering parameters are described in table 3.

Endpoint	Description
/collections	List the metadata of all available feature collections. It also allows the management of the collection configuration (i.e., create, update, and delete collections).
/collections/:cID	Retrieves the metadata of a single feature collection by its ID. Also allows the management of a feature collection by allowing insertion or deletion of the collections' data.
/collections/:cID /items	Retrieves features for a feature collection. Allows multiple filtering capabilities through URL parameters. Available filtering parameters are displayed in table 3.
/collections/:cID /items/:fID	Retrieves the data of a single feature from a feature collection.

Table 2. Endpoint description for OGC API - Features component.

3.2.4 OGC API - EDR Component: This component allows the exploration and management of EDR collections. These collections share the same characteristics as feature collections, but their underlying data and structure may be different. EDR collections can include massive datasets such as sensor observations and georeferenced timeseries which span across multiple years and have multiple instances of the same dataset. Additionally, to lower the size of EDR collections, observations may not have a location as a geometry, but as a reference to a different collection (e.g., a feature collection that contains sensor information).

Parameter	Description
datetime	Enables filtering a collection based on a specific date or a range of dates. Example: 2020-01-01/2024-01-01.
bbox	Enables filtering a collection based on a bounding box. A bounding box is a list of 2 coordinates that indicates the top-left and bottom-right corners of a rectangle. Example: 9.00,45.37,92.36,45.56
Field-specific filters	Enables filtering by specific data fields. For instance, if a collection contains a data field named "id", it would be possible to retrieve only the elements matching a specific ID by sending the parameter "id=value".

Table 3. URL Query parameters for a "/collections/:id/items" endpoint.

Given the unique characteristics of environmental data, the OGC API - EDR standard (Open Geospatial Consortium, 2023b) provides a set of endpoints that allow discovery and data querying by enabling multiple spatial query types. Each query type corresponds to an endpoint of the EDR specification. Those endpoints follow the structure *"/collections/:id/query"*, where the variable "query" can be one of the following:

- **items**: Query the EDR collection as a feature collection. This endpoint corresponds to the OGC API - Features specification.
- **position**: Query by a single position (point).
- **radius**: Query by a buffer around a single point. The units of measurement and buffer size can be specified with parameters.
- **area**: Query features within a user-defined polygon.
- **cube**: Special case of the area query, where the polygon is a 2D or 3D rectangular geometry.
- **trajectory**: Query features that lie on a linear geometry or path.
- **corridor**: Query using a linear geometry and a 3D buffer along the path.
- **locations**: Query the items of the collection by using a field that specifies the location of an object. The location is not spatial, but an identifier of a location.
- **instances**: Select, if available, one or multiple versions of the same dataset. For example, a single collection could store raw and reprocessed data in different instances.

3.2.5 OGC API - Processes Component: The Processes component in our implementation allows the creation of scripts and functions to perform operations over the collections stored on the database or over any kind of input data that is sent to the process in compliance with OGC API - Processes standard (Open Geospatial Consortium, 2021).

Processes in ROSE-API are defined as Python scripts that follow a certain structure. The structure includes metadata related

to the process (e.g., input types, output structure, version, description, title, etc.) and a function to be executed when the process is called. Discovery and execution of processes are performed by the Dynamic Processing Engine module. To manage the process execution and posterior results, this component uses a job list. A job is an abstraction of the execution of a process and contains multiple information about it, such as the status, unique identifier, and starting date and time. They are created when a process is executed and stored on the database.

Process execution could take seconds for simple operations, but may take a long time to finish, depending on the amount of input data and the complexity of the operations to be performed. According to the OGC API - Processes standard, to overcome this issue, process execution may be synchronous or asynchronous. A synchronous execution will wait until the process execution is over for the server to return a result. An asynchronous execution will run in the background and return the unique identifier of the job that was created with the process execution. When the execution is completed, the result is saved to a file on disk and is made available through the job results. This type of execution does not block the server and is more suitable for complex processes.

Independently of the execution type (i.e., asynchronous or synchronous), each time a process is executed a new job is created on the database. After the execution finishes, the result is persisted on disk, and the job saves the location of the resulting file as the execution result. Results are not saved to the database, as they constitute a document and are not suitable for being stored on a relational database. Additionally, process results could generate big files and clutter the database. When job results are requested, the server sends the contents of the results file. The endpoints provided by the Processes component are described in table 4.

The processes component allows the API to have processing capabilities. The creation of ad-hoc Python functions enables the extension of the API, as well as the usage of the underlying data through the ORM and other utility functions.

Endpoint	Description
`/processes`	Lists basic metadata for every process available on the server. The list of processes is dynamically generated based on the user-defined Python scripts.
`/processes /:pId`	Displays the complete metadata for a specific process by its ID. Complete metadata includes the input and output parameter schemas. The ID of a process is specified on the process Python script as metadata.
`/processes /:pId/ execution`	Executes a process specified by its ID. To execute a process, the Dynamic Processing Engine loads the Python script as a module and executes its main method. Execution can be synchronous or asynchronous. When a process gets executed, a job is created on the database, and when it finishes, the results are stored in a file.
`/jobs`	Lists all the jobs that have been executed by the server.
`/jobs/:jId`	Displays information of a specific job, identified by its unique identifier. Information includes its status, starting date and time, progress, process type, etc.
`/jobs/:jId /results`	Displays the result of the process execution associated with the job identified by a unique ID. This endpoint forwards the contents of the result file associated with the job as the response. In case the results are not yet available, this endpoint responds with an error.

Table 4. Endpoint description for OGC API - Processes Component.

4. Case Study

To showcase the capabilities of ROSE-API for the exposure and management of environmental monitoring data we downloaded the catalogue of air quality monitoring data of ARPA Lombardia and ingested it into an instance of ROSE-API. ARPA Lombardia is the environmental authority of the Lombardy region in Northern Italy (Regional Agency for Environmental Protection (ARPA Lombardia), 2024). This organisation maintains an environmental monitoring network of meteorological and air quality sensors. The monitored pollutants are NOX, SO2, CO, O3, PM10, PM2.5 and benzene, the number of active stations is 87. A single station may contain one or more sensors which measure a single pollutant.

The datasets that we included into ROSE-API for this case study are the list of air quality monitoring stations of the entire region, and the observations of those sensors. The dataset with air quality monitoring stations includes multiple metadata of each station, such as its location, altitude, and pollutant type. The sensor observations are provided in multiple datasets that are organized by year ranges. For instance, observations are organized in one dataset for observations from 2000 to 2009, one from 2010 to 2017, and one from 2018 to the present year.

The observations do not contain location information, but the ID of the sensor that took the measurement is provided as one of the data fields. Then, it is possible to reconstruct the location of the observation by extracting the sensor information from the air quality monitoring stations dataset. This dataset organization is useful for maintaining and organizing big amounts of data but falls short for performing queries that span across dates in multiple datasets. For instance, to query information for years in two different time ranges it is necessary to download or query two different datasets and then join the results. It is also not possible to perform spatial queries directly on the sensor observations due to the lack of location information or capabilities to extract the location of a related dataset using a single query.

These two limitations were addressed using ROSE-API. For this, we created two collections, one for the sensor information called “airqualitysensor” and a second collection for storing the observations called “airqualitymeasurement”. Figure 2 shows the database representation of both collections and their data fields. The air quality sensor collection (“airqualitysensor”) is a feature collection and contains all the information from the sensors, including its location. The air quality observations collection (“airqualitymeasurement”) is an EDR collection and contains the observations from the air quality monitoring network. It includes the measured value, the date and time of the measurement and the ID of the sensor that took the measurement. This ID is connected to the air quality sensor collection for extracting additional information about the measurement. This relationship allows to perform spatial queries directly on the air quality measurements collection.

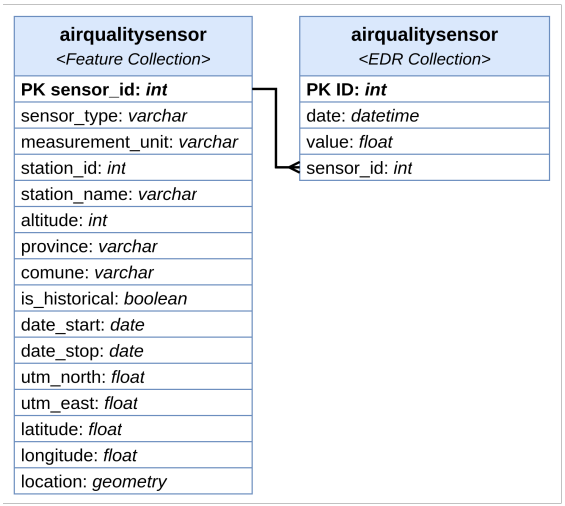

Figure 2. Entity relationship diagram of the ROSE-API collections with the air quality data from ARPA Lombardia.

We downloaded and processed the complete dataset of air quality monitoring stations and their observations from the year 2010 to the year 2023. The observations are located in two different datasets provided by ARPA Lombardia (i.e., 2010 to 2017 and 2018 to the present year). The number of air quality monitoring sensors ingested by ROSE-API is 982, active and historical, while the number of observations is 55.148.502. Air quality data is sampled with an hourly frequency, meaning that, approximately, a single sensor produces 24 data points per day. A map showing the position of the monitoring stations is displayed in figure 3.

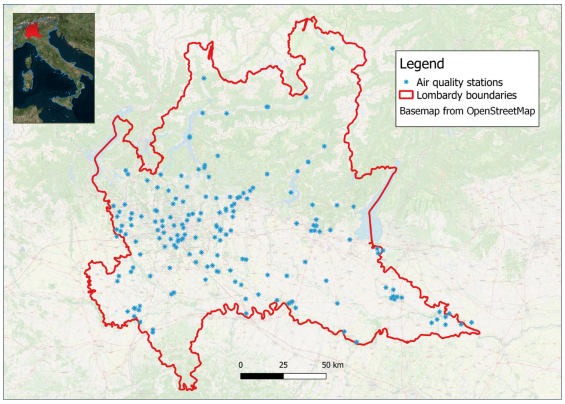

Figure 3. ARPA Lombardia air quality monitoring station locations.

ROSE-API provides processing capabilities through OGC API - Processes. To extend the default functionalities of the API, we implemented a set of functions for enabling data preprocessing directly on the server. It includes outlier removal and data aggregation functionalities.

As ARPA Lombardia data is widely used for research within the Geolab research group at Politecnico di Milano (Cedeno-Jimenez et al., 2023, Puche et al., 2023), by using ROSE-API for maintaining a local copy of ARPA data we can improve the research process using the previously mentioned data preprocessing capabilities such as outlier removal, data aggregation per unit of time (i.e., day, week, month, and year) and/or space, and powerful querying capabilities.

4.1 Results

We compared some of the data discovery and retrieval capabilities of ROSE-API with respect to the API provided by ARPA Lombardia through their Open Data Portal (Open Data Lombardia, 2023). ARPA's API is provided through an open data platform named Socrata (Tyler Technologies, n.d.). It provides a complete data management platform and a fully-featured API with discovery and filtering capabilities. Our implementation excels at the following:

- **Querying**: Both APIs can perform simple and advanced queries, however, ROSE-API is capable of doing spatial queries directly on the observations through the OGC API - EDR endpoints that are described in section 3.2.4, e.g., filter by buffer, bounding box, etc. Also, it allows complex querying based on properties of related datasets.
- **Response metadata**: ROSE-API provides multiple metadata information per response, including web linking (e.g., URL to parent collection, next and previous page URLs, alternative formats), number of matched and retrieved records, and the timestamp of the request.
- **Processing capabilities**: Through OGC API - Processes (see section 3.2.5) it is possible to extend the capabilities of the API. We created processes for data cleanup and aggregation based on multiple spatial and non-spatial filters.

However, it falls short in the following:

- **Response times**: ROSE-API is slower in comparison to ARPA's API. We performed a benchmark for calculating response times in both APIs by doing 50 requests for multiple limit sizes.

We conducted a series of tests for the querying capabilities, response metadata, processing capabilities, and response times, which are described in the following sections.

4.1.1 Querying: Both APIs can perform simple queries where filtering parameters are within the observation properties (e.g., filtering by a date range or observation value). Nonetheless, ROSE-API can perform requests for filtering observations with respect to related data, such as the location of the observation, in a single request. This is possible due to the ability of ROSE-API to interconnect collections. For the API of ARPA it is still possible to perform those types of queries, but it requires multiple steps. It is first necessary to query the sensors' dataset, extract the sensor IDs given a specific filter, and then filter the observations' dataset by that list of sensors.

A set of queries were performed for each of the APIs on the datasets containing the air quality observations. We tested the possibility of retrieving a specific set of observations using only API requests for both services, and also the possibility to retrieve the same data using a single API request. The results are reported in table 5.

Due to the capability of relating collections of ROSE-API, it is possible to perform complex queries in a single request, harnessing the power of relational databases. ARPA's API is still able to perform complex queries but requires additional steps and user processing.

Query	AP	AS	RP	RS
Retrieve observations for the entire year 2018	X	X	X	X
Retrieve observations from 01/06/2017 to 01/06/2018	X	X	X	X
Retrieve observations for the entire year 2018 using a bounding box.	X	X	X	X
Retrieve observations for the entire year 2018 for the pollutant PM10.	X	X	X	X
Retrieve valid observations (different than -9999).	X	X	X	X
Observations as GeoJSON format without geometry.	X	X	X	X
Observations as GeoJSON format with geometry.	X	X	X	X

Table 5. Ability to perform complex queries by the ARPA API and ROSE-API. The columns stand for ARPA Possible (AP), ARPA Single (AS), ROSE Possible (RP), ROSE Single (RS).

4.1.2 Response metadata: ROSE-API returns certain metadata in each response in compliance to the specifications of OGC API standards. Multiple links are provided in each response for facilitating navigation within the API (such as the URLs of the parent collection and alternative formats), and for pagination, providing the URL of the next and previous pages based on the limit and offset parameters of the current request.

In addition, ROSE-API provides the total number of records matched by a query for facilitating pagination. An example of the metadata contained in a ROSE-API response is shown in figure 4. The property "numberMatched" represents the total number of records that fulfil the query filters, while the property "numberReturned" represents the number of records that are present in the response. An example of a "next" link is also displayed, where the "limit" and "offset" parameters control the pagination of results.


```
{
  ...
  "numberMatched": 2787175,
  "numberReturned": 1000,
  ...

  "links": [
    {
      "href": "{{Base URL}}/items?limit=1000&offset=1000",
      "rel": "next",
      "type": "application/geo+json",
      "title": "Next page"
    }
  ]
}
```


Figure 4. Example of response metadata in ROSE-API.

None of these metadata is returned by the responses of ARPA's API. To see the total number of records matched by a filter it would be necessary to perform an additional request counting the result records. Then, pagination is managed by the user by using the page size (limit) and the total number of records.

4.1.3 Processing capabilities: To provide additional meaning to data, processing is necessary. By expanding the basic capabilities of an API it is possible to improve the data processing. ROSE-API allows the expansion of the discovery and querying capabilities of the API by offering the possibility to create ad-hoc functionalities using Python scripts. We created an ad-hoc process for data aggregation that performs the operations directly on the database. This process allows the data aggregation from the "airqualitymeasurements" collection by year, month, or date, as well as applying spatial and non-spatial filters.

As an example, we retrieved the monthly aggregation of PM10 observations for sensors located in the city of Milan using a single request to the API. The result consists of a list of 527 observations corresponding to the monthly average PM10 concentration for 6 different sensors over 9 years. Note that the total number of records should correspond to 648, however, not all sensors recorded observations for the whole 9 years. Figure 5 shows a time-series plot of the aggregated data.

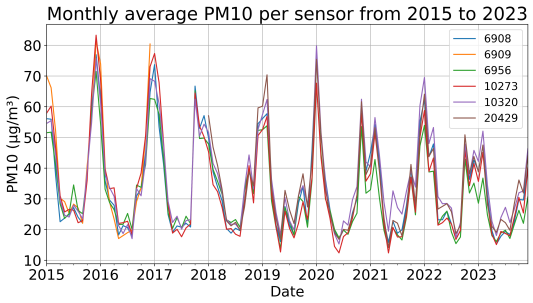

Figure 5. Plot of aggregated monthly average PM10 concentration per sensor in Milan from 2015 to 2023 using data processed by ROSE-API.

To perform such an aggregation using ARPA's API it would be necessary to perform a spatial query to the air quality sensors dataset to retrieve the sensor IDs of PM10 sensors located in Milan. Then, extract the data from the datasets of air quality observations of 2010 to 2017 and of 2018 onwards using the sensor IDs as filters. Finally, the downloaded data would have to be combined and processed locally.

4.1.4 Response times: Response times for ARPA's API are better with respect to our implementation. A test was carried out where both APIs were queried 50 times requesting an increasing number of features (1000, 10000, 100000). The results are reported in figure 6.

In general, ARPA responses are faster than ROSE-API, but for larger requests, the standard deviation is larger in ARPA. This means that response times for large requests are not stable, and could end up taking more time than expected, even surpassing the ROSE-API responses. The standard deviation for ROSE-API is smaller and the variability of response times among requests of the same size is very small. Nevertheless, the difference in response times between both APIs is less than 1 second for each request size.

4.2 Compliance Tests

For an implementation of any OGC standard, there is a set of requirements that must be fulfilled. The Compliance Interoperability & Testing Evaluation (CITE) is a subsection of OGC that creates and maintains the test sets to be performed to an implementation of a standard. To test the compliance of ROSE-API, we tested the endpoints of the Features and EDR components.

ROSE-API was tested against part 1 (core) of the OGC API - Features standard and against part 1 of the OGC API - EDR standard. The tests were performed using the ROSE-API instance with the ARPA Lombardia data used for the case study.

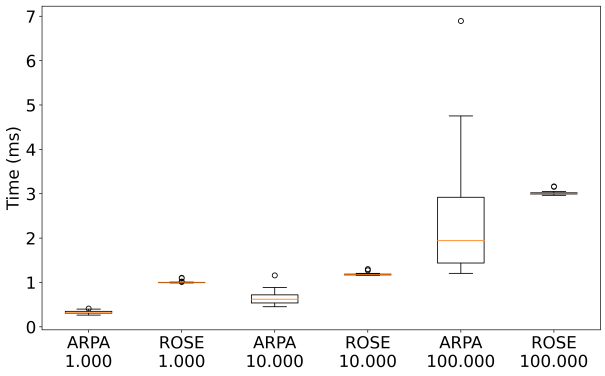

Figure 6. Box plot of response time for multiple requests sizes.

Compliance testing was performed using the testing suites provided by OGC.

For OGC API - Features, all the tested endpoints passed. Tests included validation of the OGC API - Common interfaces (OpenAPI document, compliance, and landing page), collection metadata validation, and query parameters available for each endpoint. For OGC API - EDR, all the tested endpoints passed. The corridor test was skipped as it was not implemented, and the cube endpoint test failed due to a known issue with the testing suite. EDR tests included JSON and GeoJSON format validation, query parameters for each spatial query type, and collection metadata.

5. Conclusion

Geospatial standards are necessary for interoperability which supports environmental data monitoring and management. The OGC has recently proposed the OGC API standards which provide specifications for environmental data discovery, querying, and processing.

The adoption of such standards by environmental organisations is key for ensuring interoperability, however, their implementation requires an effort. We propose ROSE-API (Reusable Open Source Environmental-Data-Management OGC API), a web service which is able to ingest environmental data and expose it using the OGC API standards, i.e., features, EDR, and processes. The proposed API is dynamic and modular. It dynamically generates the collections based on the database tables on runtime, allowing the querying of data. Ad-hoc data processes can be added in a modular way following a specific structure to be interpreted by the server. Finally, the OpenAPI document is dynamically generated, providing the endpoint descriptions for the mentioned collections and processes.

As a case study to showcase ROSE-API's capabilities for the discovery and query of environmental data, we ingested air quality monitoring data into an instance of ROSE-API. The data consists of hourly concentrations of 7 pollutants in 87 stations from 2010 to 2023 in the Lombardy region, Italy. ROSE-API allows to perform simple and advanced queries in addition to spatial queries and geometry retrieval. The API responses returned also specific metadata useful for pagination. Furthermore, ad-hoc processes were created to perform the temporal and spatial aggregation of the air quality observations.

References

Blanc, N., Cannata, M., Collombin, M., Ertz, O., Giuliani, G., Ingensand, J., 2022. OGC API STATE OF PLAY - A PRACTICAL TESTBED FOR THE NATIONAL SPATIAL DATA INFRASTRUCTURE IN SWITZERLAND. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 48, 59 - 65. doi.org/10.5194/isprs-archives-XLVIII-4-W1-2022-59-2022.

Cedeno-Jimenez, J., Pugliese-Viloria, A., Brovelli, M. A., 2023. Estimating Daily NO2 Ground Level Concentrations Using Sentinel-5P and Ground Sensor Meteorological Measurements. *ISPRS International Journal of Geo-Information 2023, Vol. 12, Page 107*, 12, 107. doi.org/10.3390/ijgi12030107.

Environment and Climate Change Canada (ECCC), 2020. MSC GeoMet API and geospatial web services. https://www.canada.ca/en/environment-climate-change/services/weather-general-tools-resources/weather-tools-specialized-data/msc-geomet-api-geospatial-web-services.html (21 September 2020).

Geonovum, 2021. Geonovum ogc api testbed - ogc-api-testbed documentation. https://apitestdocs.geonovum.nl/ (2021).

Open Data Lombardia, 2023. ARPA Lombardia — Open Data Lombardia. https://www.dati.lombardia.it/.

Open Geospatial Consortium, 2021. OGC API - Processes - Part 1: Core. http://www.opengis.net/doc/IS/ogcapi-processes-1/1.0 (20 December 2021).

Open Geospatial Consortium, 2022. OGC API - Features - Part 1: Core. http://www.opengis.net/doc/IS/ogcapi-features-1/1.0.1 (11 May 2022).

Open Geospatial Consortium, 2023a. OGC API - Common - Part 1: Core. http://www.opengis.net/doc/is/ogcapi-common-1/1.0 (28 March 2023).

Open Geospatial Consortium, 2023b. OGC API - Environmental Data Retrieval Standard. http://www.opengis.net/doc/IS/ogcapi-edr-1/1.1 (17 July 2023).

Puche, M., Vavassori, A., Brovelli, M. A., 2023. Insights into the Effect of Urban Morphology and Land Cover on Land Surface and Air Temperatures in the Metropolitan City of Milan (Italy) Using Satellite Imagery and In Situ Measurements. *Remote Sensing 2023, Vol. 15, Page 733*, 15, 733. doi.org/10.3390/rs15030733.

Regional Agency for Environmental Protection (ARPA Lombardia), 2024. ARPA Lombardia — Home. https://www.arpalombardia.it/.

Santana, I. L., Davis, C. A., 2023. Comparative Performance Evaluation of OGC API and OGC Web Feature Service. *Proceedings of the Brazilian Symposium on GeoInformatics*, 134 - 143.

Tyler Technologies, n.d. Socrata Developers — Socrata. https://dev.socrata.com/.

United Nations Economic Commission for Europe (UNECE), 2003. ENVIRONMENTAL MONITORING AND REPORTING. https://unece.org/sites/default/files/2020-12/Final.English.Rev.2.01.12.05.*pdf* (2003).