Low-Code Framework for IoT Data Warehousing and Visualization

Victor Lamas[a], Alejandro Cortiñas[a] and Miguel R. Luaces[a]

[a]*Universidade da Coruña, Centro de Investigación CITIC, Database Lab., Elviña, 15071, A Coruña, Spain*

ARTICLE INFO

Keywords:
Internet of Things
Data Warehouse
Geographic Information System
Software Product Line
Domain-Specific Language

ABSTRACT

Background: The Internet of Things has revolutionized data collection in geosciences through extensive sensor networks. However, developing web-based data warehousing systems for IoT data remains costly and complex. While studies address sensor variability and data ingestion architectures, they often overlook the critical data warehouse component needed to manage IoT data volume and variability. Additionally, Model-Driven Engineering techniques have been used to create dashboards for urban activities but lack advanced map-based visualizations, which are essential for geospatial data. **Objectives:** This study aims to address the challenges of creating IoT data warehouses for geosciences, encouraging scientists to share sensor data analysis results using a simple, user-friendly, and cost-effective approach. **Methods:** The proposed framework integrates i) a Domain-Specific Language metamodel to define sensors, dimensions, and measurement parameters, ii) a Software Product Line for IoT data warehouse creation, and iii) a low-code platform with command-line and web interfaces. The approach was validated through four case studies: meteorological, traffic and air quality, coastal, and oceanic monitoring systems. **Results:** The framework enables efficient IoT data warehouse creation with customized spatial, temporal, and attribute aggregation. Case studies demonstrate adaptability across domains, supporting real-time data ingestion, sensor mobility, and advanced visualization. **Conclusion:** The study presents a scalable, user-friendly framework for IoT data warehousing in geosciences using SPL and DSL technologies, addressing domain-specific challenges and empowering non-expert users. Future work includes usability assessments and expansion to other domains.

CRediT authorship contribution statement

Victor Lamas: Conceptualization, Investigation, Development, Writing - Original Draft, Writing - Review & Editing. **Alejandro Cortiñas:** Conceptualization, Validation, Methodology, Supervision, Writing - Original Draft, Writing - Review & Editing. **Miguel R. Luaces:** Conceptualization, Formal Analysis, Funding acquisition, Methodology, Project administration, Validation, Resources, Supervision, Writing - Original Draft, Writing - Review & Editing.

1. Introduction

The Internet of Things (IoT) has transformed geoscience data collection by integrating vast sensor data to understand Earth's processes. IoT systems with geographic data are widely used in disciplines like environmental sciences, atmospheric sciences, and oceanography to monitor weather, air quality, and ocean parameters with specialized sensors and real-time data. To make sense of this wealth of information, researchers require tools to collect, explore, filter, and aggregate data across temporal, spatial, and attribute dimensions. Effective data collection ensures the reliability of datasets, while exploration and filtering help identify patterns, reduce noise, and refine information for analysis. Aggregating data by time, space, and attributes enables trend analysis, anomaly detection, and the discovery of meaningful correlations, ultimately supporting more informed decision-making.

Developing IoT data warehouses for geoscience is challenging due to real-time data ingestion requiring stream processing and robust databases for querying sensor and measurement data (Chan and Ueda (2000)). The system must

ORCID(s): 0000-0001-8960-1299 (V. Lamas); 0000-0002-2555-6342 (A. Cortiñas); 0000-0003-0549-2000 (M.R. Luaces) efficiently perform spatial and temporal aggregation of millions of measurements with low response times and display filtered, aggregated data on interactive maps. Alternatives like ArcGIS[1] and Carto[2] offer advanced features for managing raster and image data, but they are not ideally suited for handling sensor data. These commercial products, although powerful, do not align well with the specific needs of geoscience IoT data warehouses. We believe it is important to highlight the absence of an open-source alternative with similar capabilities, as this motivates the development of our solution. Our approach aims to fill this gap, offering a more flexible, open, and scalable alternative to the limitations of current tools.

Although each IoT data warehouse is unique, they share common features: i) stream-oriented sensor data ingestion, ii) time-series data storage, iii) tools for data discovery and exploration, iv) processing capabilities for generating insights, and v) GIS-based tools for map visualization alongside traditional listings. This suggests they can be managed as a software product family using variability management strategies. We introduce a free, open-source framework for building sensor-based, web-centric data warehousing systems using software product line (SPL) technology. Our contribution focuses on three key areas: i) a Domain-Specific Language (DSL) for defining IoT data warehouses based on sensors, measurements, and dimensions, ii) an SPL for creating information systems to collect, store, and visualize IoT data, and iii) a Low-Code Development Tool for building advanced sensor-based systems via a web browser or command line.

These capabilities are particularly crucial in machine learning-based engineering surveying, where structured and well-processed data serve as the foundation for predictive modeling and automated analysis. Some illustrative examples include Ortiz and Zamudio-García (2025), where AI-driven water management leverages real-time IoT data and machine learning models to optimize water distribution, detect leaks, and ensure water quality for sustainable infrastructure and public health. Similarly, Chaves et al. (2025) explores the use of soft sensors and streaming data analytics, powered by Kafka-ML, for real-time prediction of nitrate concentrations in aquifers, enhancing cost-effective water resource monitoring. Additionally, Sreyas et al. (2025) demonstrates how IoT-integrated sensor networks and machine learning models analyze pollution patterns, issue real-time alerts, and generate short- and long-term air quality forecasts to support environmental and public health initiatives.

Building a robust sustainable public health surveillance infrastructure demands similar sophisticated data handling. A DSL that allows for the precise definition of sensors and measurement parameters is essential, enabling the capture of critical health-related data from diverse sources, such as air quality sensors (measuring PM2.5), water quality probes (detecting pathogens), and even mobile devices (tracking anonymized mobility). Furthermore, the ability to define spatial, temporal, and categorical dimensions is vital; the spatial dimension allows for mapping disease clusters and identifying high-risk areas, the temporal dimension enables tracking disease trends and detecting outbreaks, and the

[1] `https://arcgis.com/`
[2] `https://carto.com/`

categorical dimension facilitates the analysis of health disparities across different populations. Real-time data ingestion ensures timely awareness of emerging threats, while efficient data storage and querying supports the analysis of large datasets needed for effective machine learning models. Finally, the capacity for filtering and aggregation by time and space allows for targeted investigations and the preparation of data for analysis, and advanced map-based visualizations provide powerful tools for communicating insights and informing public health interventions.

The paper is organized as follows: Section 2 discusses IoT and sensor applications across various domains, highlighting practical use cases to demonstrate the advantages of an SPL for this domain. Section 3 reviews prior research on domain-specific languages and software product lines for geographic information systems. Section 4 introduces our proposed tool and methodology. Section 5 summarizes the domains and key findings. Finally, Section 6 presents conclusions and future research directions.

2. Scenarios and Motivation

Sensor data integration and analysis are increasingly common in geoscience domains such as environmental sciences, meteorology, climatology, or oceanography (Yang et al. (2022); Chen et al. (2023); Raj et al. (2024)). IoT data warehouses in these domains share common requirements: a data model with sensors, measurements, and spatial/temporal dimensions; categorical dimensions for specific attributes; filtering and aggregation by time and space; and, in some cases, support for mobile sensors. We analyze four case studies to validate this, each reflecting distinct challenges and requirements. Each scenario illustrates the data warehouse design using a notation based on the Dimensional Fact Model (DFM), a graphical formalism tailored for the conceptual modeling phase of data warehouse projects, suitable for both analysts and non-technical users (Golfarelli et al. (1998)).

Additionally, corroborative studies, such as those by Ma (2017); Martinez et al. (2024); Silva et al. (2023); Bourgeois et al. (2018) and others, have further affirmed the statistical reliability of these principles. The adoption of a low-code development platform, as incorporated in our web and command-line interfaces, aligns with trends in simplifying the creation and deployment of data-intensive applications for non-expert users Rojas et al. (2020). The ability for geoscientists without extensive programming knowledge to define sensors and measurement parameters through our DSL directly addresses the need for domain-specific tools that abstract away technical complexities, as advocated in the context of IoT applications by Mardani Korani et al. (2023). Our framework's validation across diverse geoscience domains – meteorological monitoring, traffic and air quality, coastal, and oceanic monitoring – reflects the broad applicability and the critical role of IoT data warehousing in these fields, supported by dedicated research efforts such as those focused on IoT ocean observatory systems Raj et al. (2024) and high spatiotemporal resolution frameworks for urban temperature prediction using IoT data Yang et al. (2022). These examples, alongside the adaptability to real-time data ingestion and advanced visualizations demonstrated in our case studies, underscore the pressing need for flexible and user-friendly solutions for managing the increasing data streams in geoscience, a need further emphasized by frameworks aimed at handling big data analytics in IoT environments Vögler et al. (2017).

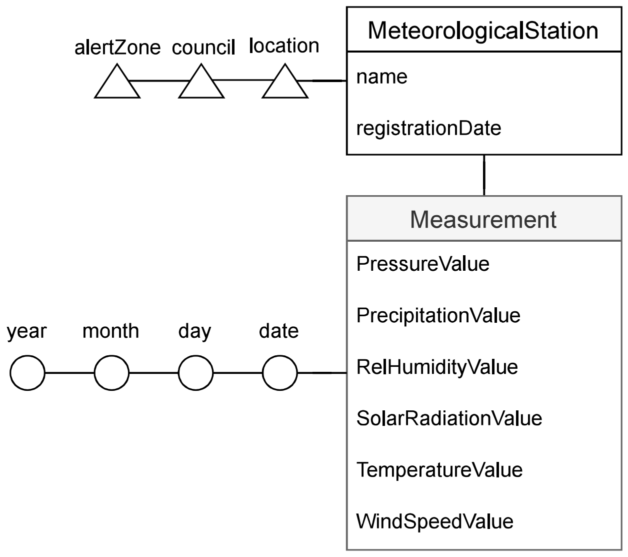


Figure 1: Dimensional Fact Model of the Meteorological Monitoring System

Meteorological Monitoring System. The data model shown in Figure 1 captures essential information about meteorological stations and their measurements. We built the data model from a dataset provided by MeteoGalicia[3], the official weather service for Galicia, Spain. The dataset can be retrieved from MeteoGalicia historical data[4]. The `MeteorologicalStation` entity represents the physical stations collecting weather data, with attributes such as `name` and `registrationDate`. The `Measurement` entity captures the observed environmental variables (e.g., `PressureValue` for the atmospheric pressure, along with the `date` attribute, which records the timestamp of each measurement). Finally, the `location-council-alertZone` spatial dimension locates the meteorological stations in a specific alert zone.

This example illustrates key requirements for an IoT data warehouse: sensors collect timestamped measurements within a hierarchical spatial structure. Enabling filtering and aggregation by time and space to identify trends and provide stakeholder summaries. The example demonstrates variability in three areas: i) the number of parameters (we selected five from over fifteen available), ii) the type of spatial hierarchy (e.g., different administrative divisions grouping stations), and iii) time granularity, ranging from minute-level precision to daily or monthly summaries based on the application.

[3]https://meteogalicia.gal/
[4]https://www.meteogalicia.gal/observacion/estacionshistorico/historico.action?request_locale=es, Spanish only

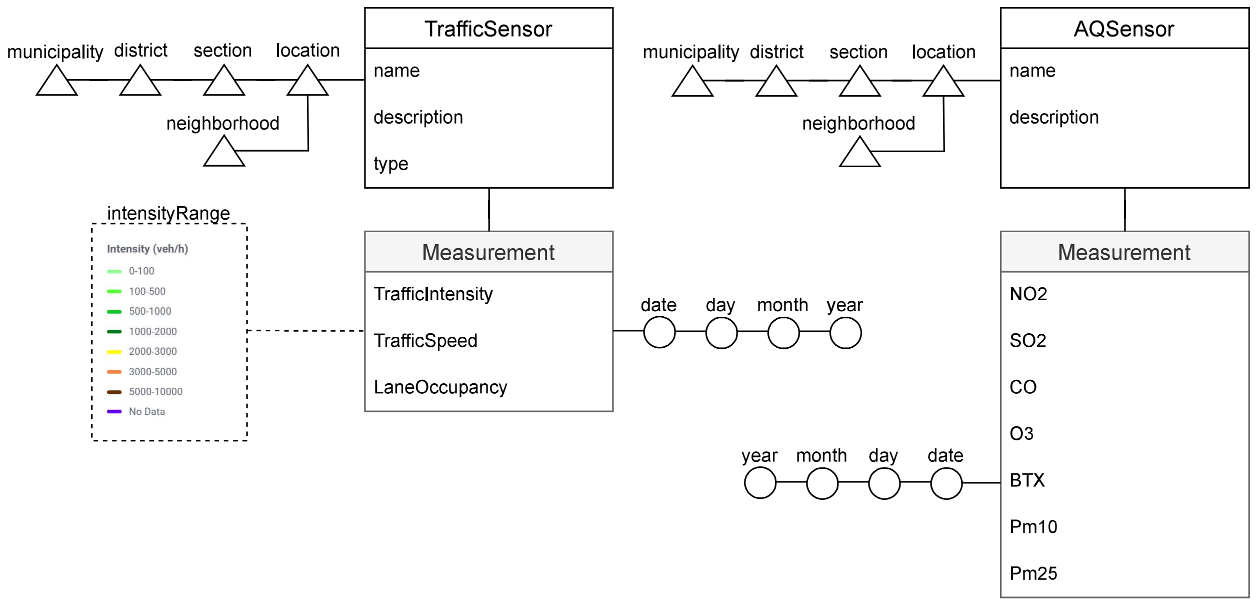


Figure 2: Dimensional Fact Model of the Traffic and Air Quality System

Traffic and Air Quality. The data model in Figure 2 captures key traffic and air quality monitoring information based on datasets from the Madrid City Council[5]. The `TrafficSensor` entity represents physical stations collecting traffic data. The `Measurement` entity logs traffic data, including `TrafficIntensity` (vehicles per unit time), `TrafficSpeed` (average speed), and `LaneOccupancy` (lane occupancy percentage), all timestamped. The `AQSensor` entity represents air quality monitoring stations. Its `Measurement` entity records environmental data (e.g., `NO2` (nitrogen dioxide), `BTX` (aromatic hydrocarbons), or particulate matter levels (`PM10`, `PM2.5`)), all tagged with timestamps. Both sensor networks share a spatial hierarchy of locations, sections, districts, and municipalities.

This example shares a commonality with the previous one (e.g., timestamped measurements, a hierarchical spatial structure, etc.). It also highlights variability in measured parameters and spatial dimensions. Additionally, it introduces a new variability: some measurements are best visualized using ranges with distinct colors for each range. To support this, the data model includes an `IntensityRange` enumeration linked to the measurements. Different IoT data warehouse dashboards may require custom-defined ranges tailored to specific application needs.

Coastal Monitoring System. The Galician Technological Institute for Marine Environment Control (INTECMAR)[6] runs a network of CTD (conductivity, temperature, depth) sensors on maritime buoys across 54 stations in five Galician estuaries. Figure 3 shows a subset of this data for a warehouse model. These sensors correspond to the `BuoySensor` entity, while `Measurement` captures marine parameters like `density`, `oxygen`, and `temperature`, each tagged with a `timestamp`.

[5]https://datos.madrid.es/portal/site/egob
[6]http://www.intecmar.gal/Intecmar/

This example shares the same commonality and variability as the previous ones. Still, it introduces a new type of variability: sometimes, specific measurements must be promoted to dimensions within the data warehouse. A clear example here is the depth at which measurements are taken. Furthermore, ranges are defined to manage their continuous nature as a double-precision value, allowing for categorization into predefined labels.

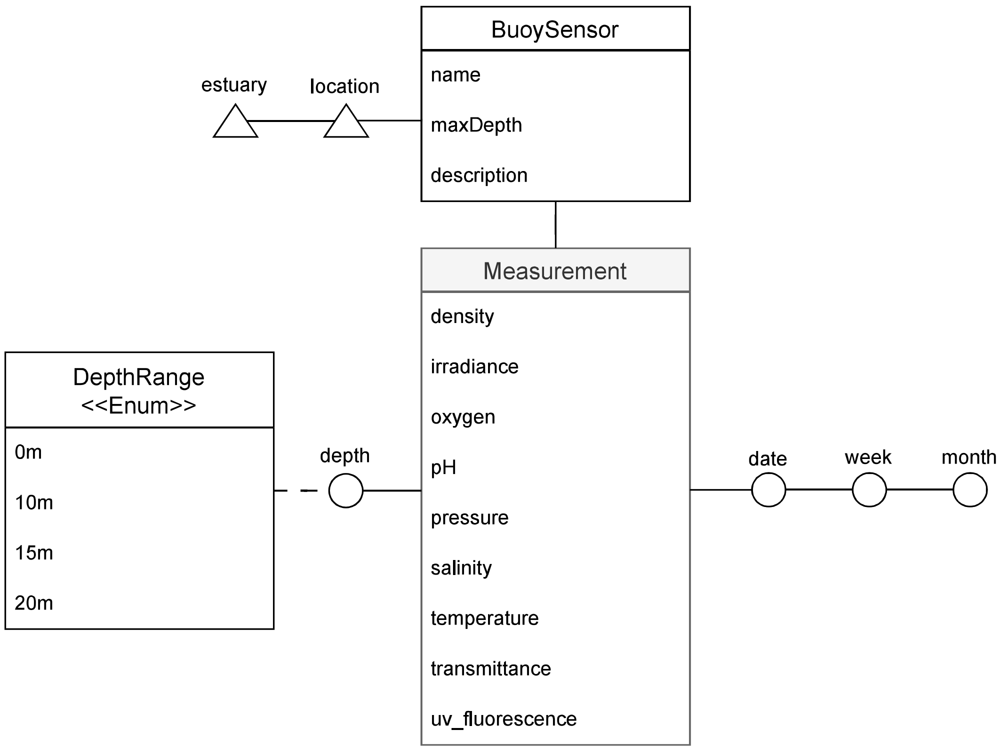


Figure 3: Dimensional Fact Model of the Coastal Monitoring System

Oceanic Monitoring System. The Argo Program[7] utilizes a global network of profiling floats and ship-mounted sensors for detailed oceanographic data. The model in Figure 4 displays an IoT data warehouse model created from the information downloaded from the Euro-Argo fleet monitoring tool[8]. The ships are represented by the `ShipMovingSensor` entity, which holds information about the ships, such as their unique name, description, and the type of ship they are mounted on. The `Measurement` entity stores the actual measurements (i.e., temperature, pressure, salinity). Similarly to the previous cases, this model includes temporal and categorical dimensions (depth). However, a new form of variability arises here: the location is associated with the measurement rather than the sensor, reflecting that the sensors are mounted on moving ships. Hence, the IoT data warehouse must support moving sensors.

[7] https://argo.ucsd.edu/
[8] https://fleetmonitoring.euro-argo.eu/dashboard

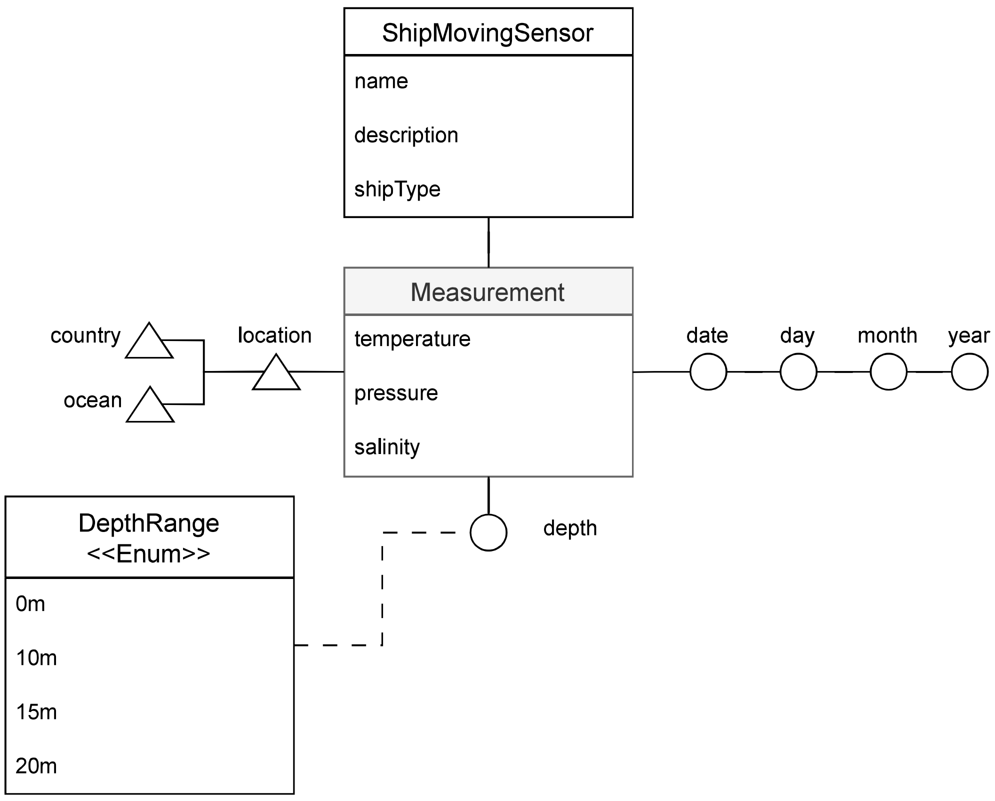


Figure 4: Dimensional Fact Model of the Oceanic Monitoring System

Motivation and research questions. From the examples discussed in the previous section, it is evident that IoT systems share common characteristics, such as the use of sensors that capture measurements with temporal and spatial dimensions, the need for data classification and querying, and the ability to visualize data through various methods like maps and charts. However, there is also variability among them, including the properties of the sensors, the parameters they measure, the dimensions created from these measurements, the need for defining ranges for certain values, the necessity of categorical dimensions for classification, and scenarios where the sensors themselves move. Finally, we must remember that geoscience researchers are not experts in computer science, and they require a simple, user-friendly tool that allows them to create IoT data warehouses without the complexity of sophisticated programming or technical knowledge. Based on these observations, we propose the following research questions:

1. **RQ1. Can we use the commonality between the domains to build a software product line that implements IoT data warehouses?** This question explores the potential to identify shared features across different IoT domains to design a unified software architecture. The goal is to create a software product line capable of developing customizable data warehouses for various IoT data types.
2. **RQ2. Can we describe the variability of the data model of IoT systems?** This question addresses the need to capture and model the diversity of IoT systems by identifying and representing their data-related variations.

3. **RQ3. Can we help researchers create and deploy systems that visualize their IoT data easily?** This question focuses on creating a user-friendly, low-code platform that enables researchers with minimal technical knowledge to design and deploy systems for visualizing their IoT data.

3. Background and related work

Model-driven engineering (MDE) is widely used to integrate or migrate technologies, ensuring compatibility, supporting maintenance, and documentation (Mardani Korani et al. (2023)). Studies have formally defined IoT sensor variability (Chaudhary et al. (2022)) and described architectures for ingesting sensor data, addressing big data challenges (Vögler et al. (2017); Pham et al. (2015); Kefalakis et al. (2019)), and handling variability across data sources (Erraissi and Belangour (2018)). Although these systems propose solutions to handle variability in the specification of sensors and the architectures, they do not address the challenges associated with the data warehouse component. This limitation highlights the need for further research to integrate data warehousing solutions that can effectively manage and utilize the variability and volume of IoT-generated data. Another key finding is the demand from citizens and governments for dashboards visualizing urban activities. Some state-of-the-art MDE techniques use domain-specific languages to define city sensors and create dashboards (Erazo-Garzon et al. (2023); Rojas et al. (2020)), and some proposals model smart cities focusing on infrastructure and analytics (Basciani et al. (2020)). While some systems support dashboards, they lack advanced map-based visualizations crucial for geographic sensor data. This need spans urban and non-urban settings, such as seawater or ship monitoring, where geoscience experts would benefit from systems utilizing geographic information across domains.

This work builds upon three previous contributions. In Alvarado et al. (2020), a DSL for web-based GIS applications was introduced, enabling users to define visualization models declaratively, including maps, layers, styles, and entity specifications. We use this DSL internally to describe the GIS framework supporting the IoT data warehouse. In Lamas Sardiña et al. (2024), GisPublisher was presented as a tool for simplifying GIS visualization creation using shapefiles, allowing researchers to focus on data rather than development. Built on the SPL and DSL from Alvarado et al. (2020), it includes a command-line interface for deploying GIS systems. In Cortiñas et al. (2024), we present our first proposal to build SensorPublisher. That work primarily focused on the software product line aspects, providing a general framework for sensor data processing and visualization. However, it did not describe the system's architecture or workflow in detail. This paper extends it by offering a comprehensive discussion of these aspects, supporting moving sensors, and shifting the focus toward the applicability of the system in specific sectors. Additionally, the software has undergone significant improvements, including refinements to its architecture and the replacement of Elasticsearch with TimescaleDB to enhance performance. Finally, this article provides a thorough validation of the system by evaluating it against real-world constraints and requirements.

4. Proposal

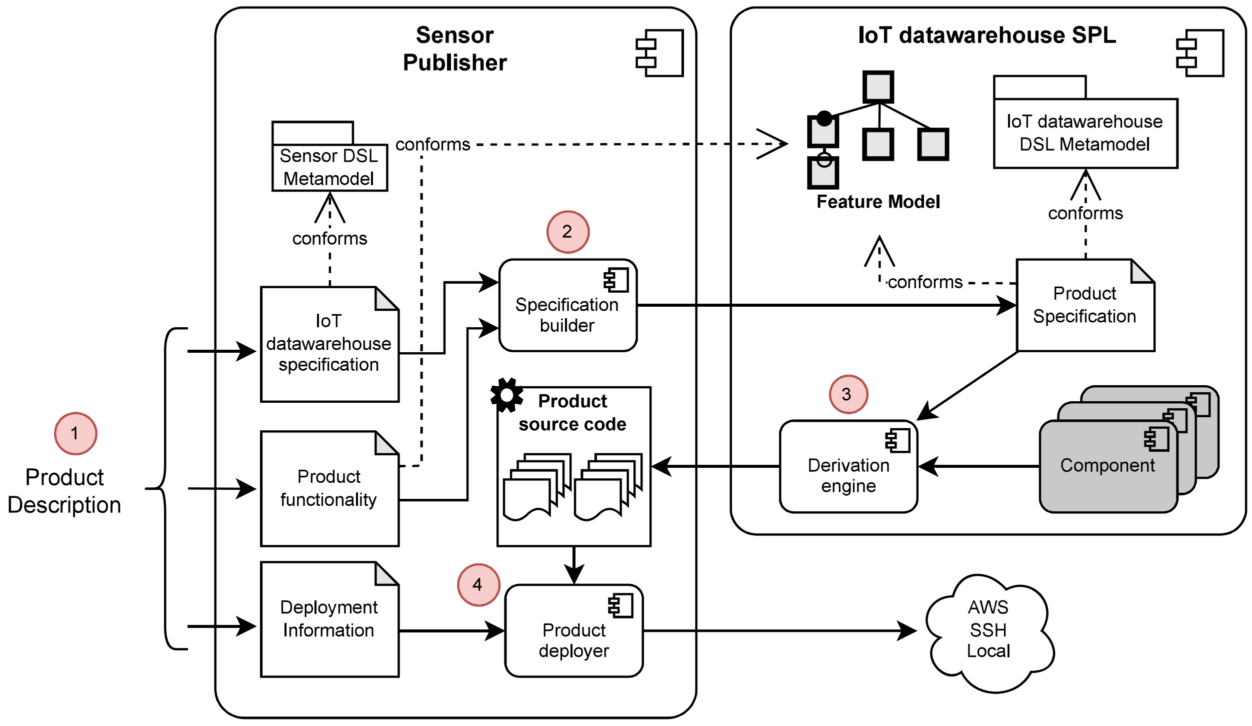


Figure 5: Framework architecture

Figure 5 shows the architecture and workflow of the proposed framework, composed of the Sensor Publisher component on the left and the IoT data warehouse SPL. The process begins (step 1) with the user providing a detailed product description, including the IoT data warehouse specification (conforming to the Sensor DSL Metamodel in Section 4.1), selecting desired features from the IoT data warehouse SPL feature model, and specifying deployment details. The Specification Builder then combines these inputs to generate a product specification (step 2), ensuring that the feature selection adheres to the feature model constraints and that the user-provided sensor definitions are accurate. The output specification conforms to the IoT data warehouse DSL Metamodel (Section 4.2). The Derivation Engine uses this specification to assemble the product source code integrating the appropriate SPL components as described in Section 4.3 (step 3). The Product Deployer handles deployment (step 4), supporting local, remote (via SSH), and cloud environments like AWS. A web-based and command line tool, described in Section 4.4, simplifies this process for researchers and facilitates integration into deployment pipelines.

4.1. Sensor DSL

A Domain-Specific Language (DSL) is a programming language or specification dedicated to a specific problem domain, designed to concisely express that domain's concepts and structures (Fowler (2010)). In contrast to general-

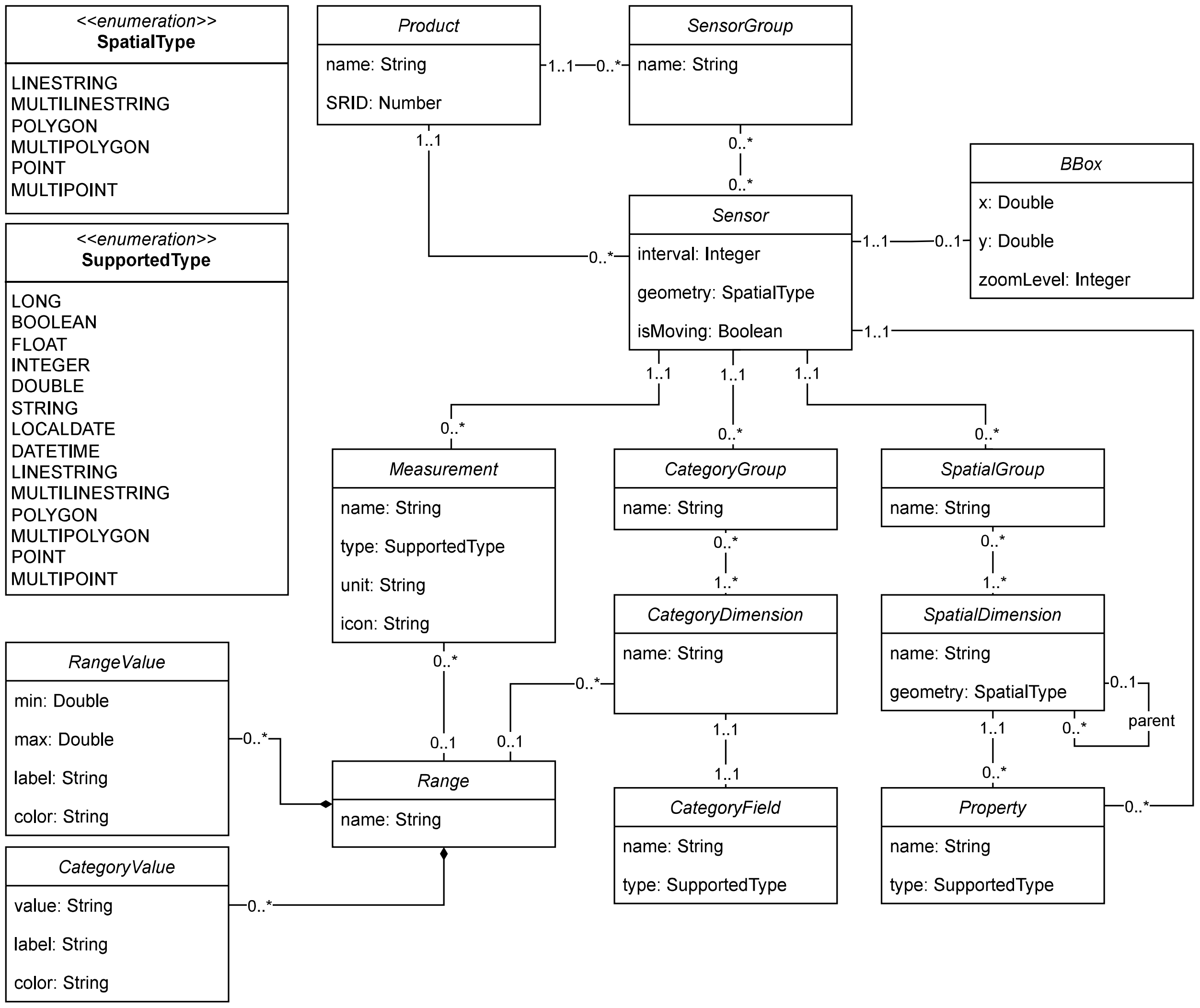


Figure 6: Class Diagram of the DSL Metamodel

purpose programming languages, DSLs focus on a narrow scope, often leading to increased productivity and easier readability within that specific context.

Figure 6 shows the Sensor DSL metamodel. Our metamodel comprises two fundamental classes: `Sensor` and `Measurement`. The class `Sensor` represents a group of sensors of a particular type. Each sensor type has a name, a data collection interval that specifies how frequently the sensor will collect data, a kind of sensor geometry (e.g., point, polygon from the enumeration `SpatialType`), and a boolean value (`isMoving`) to specify whether the sensor is static or moves over time. While our system supports sensors with geometries of any type, the specific choice of geometry (point, line, or polygon) does not significantly impact the core functionality. However, in some applications where spatial context is crucial, more complex geometries may have visual relevance, and we provide this flexibility without introducing substantial computational overhead. Each sensor is associated with a set of properties represented by the

Property class. A property is defined by its name and data type, where the data type can be any of those listed in the enumeration SupportedType (e.g., STRING, DOUBLE, DATETIME). To enable the visualization of all sensors on a map with a fixed initial position, each Sensor may include a bounding box, defined by the BBox class, which specifies the x and y coordinates and an initial zoom level. The Measurement class specifies the measurements that the sensor collects over time. Each measurement is defined by a name and data type. Additionally, measurements can specify their units (e.g., °C, m/s) and an icon that represents them visually, which can be any of the Material Design Icons[9]. This structure supports sensor variability by allowing each sensor type to define its unique geometry, collection interval, and customizable properties. Variability in measurements is supported by enabling each measurement to define its unique name, data type, units, and visualization attributes. For enhanced data visualization, measurements can include a Range, which defines how data is grouped or visualized. A range can be a set of intervals (each with a minimum, maximum, label, and color) or discrete values.

To construct a data warehouse using sensor data, each Sensor and its associated Measurements are treated as a data warehouse fact table. The metamodel supports defining spatial and categorical dimensions for sensors. Spatial dimensions, represented by the SpatialDimension class, have a name, a geometry type (from the SpatialType enumeration), and properties. These dimensions can form hierarchical relationships through a parent-child link, enabling filters and aggregations in spatial analyses. This ensures variability in spatial dimensions by allowing flexible hierarchies and customizable geometry types and properties. Categorical dimensions, represented by the CategoryDimension class, are defined by a name and a field specifying the attribute that the final data warehouse will use to represent categorical values. Each sensor connects to its spatial and categorical dimensions through SpatialGroup and CategoryGroup, respectively. Temporal dimensions are automatically derived from the interval attribute of each sensor, eliminating the need for explicit user specification. To organize sensors, the SensorGroup class allows grouping sensors into collections. Finally, the Product class serves as the top-level representation of the entire project, providing the system's name and spatial reference system identifier (SRID). This enables the specification of the coordinate reference system used throughout the project.

The following listings demonstrate the textual syntax of our DSL. CREATE PRODUCT is the first sentence in a product specification (see Listing 1), and it is used to define the product name and the SRID. The sentence CREATE [MOVING] SENSOR can then be used to specify each sensor definition (see Listing 2). Each type of sensor can have its properties (in the WITH PROPERTIES clause), measurements (in the MEASUREMENT DATA clause), spatial dimensions (in the SPATIAL GROUP clause), categorical dimensions (in the CATEGORICAL GROUP clause), sensor network bounding box (in the BBOX clause), and whether the sensor will always be in the same place or move (the optional keyword MOVING). To enable the end user to aggregate and filter by these ranges, a RANGE can also be added to CATEGORICAL

[9]https://m3.material.io/styles/icons/overview

GROUP or measurements.

```
CREATE PRODUCT productName USING srid;
```

Listing 1: CREATE PRODUCT sentence

```
CREATE [MOVING] SENSOR <sensorName> (
    interval: integer,
    geometry: geometryType
) WITH PROPERTIES (
    <propertyName> dataType [DISPLAY_STRING] [REQUIRED] [UNIQUE],
    ...
) WITH MEASUREMENT DATA
    <measurementName> dataType [UNITS String] [ICON String] [RANGE rangeRef]
    ...
) WITH SPATIAL GROUP <dimAggNameGroup> (
    <spatialDimRef1>,
    <spatialDimRef2>,
    ...
) WITH CATEGORICAL GROUP <catAggNameGroup> (
    <catDimRef1> [RANGE <rangeRef1>],
    ...
) WITH BBOX
    ([Double, Double], Integer) | (Double, Double, Integer)
);
```

Listing 2: CREATE SENSOR sentence

To support reusability, spatial and categorical dimensions and ranges are defined with specific sentences. Listing 3 displays the CREATE SPATIAL DIMENSION sentence's syntax. In the same way, Listing 5 displays the CREATE CATEGORICAL DIMENSION statement's syntax.

```
CREATE SPATIAL DIMENSION <dimNameRef> (
    geometry: geometryType
) WITH PROPERTIES (
    <propName> dataType1 [DISPLAY_STRING]
    ...
) WITH PARENT (
    <dimNameParentRef>
);
```

Listing 3: CREATE SPATIAL DIMENSION sentence

As can be seen in Figure 6, a `Sensor` has a set of `Spatial Group` associated with it, which in turn have a set of `Spatial dimension` associated with it; furthermore, these `Spatial Dimension` can be associated with other `Spatial Dimensions` so that they are represented as a hierarchy. As seen in the scenarios Section 2, it is common for spatial dimensions to have a hierarchy, such as section, district and municipality, so that the administrative separations within a city are represented. The keyword `WITH PARENT` is used to represent this hierarchy in the DSL.

Different `SPATIAL DIMENSIONS` can be grouped within `DIMENSION GROUPS`; this helps users define sets of hierarchies or spatial dimensions under different groups, making it easier in end application views to perform queries on spatial aggregations, being able to select the group of spatial dimensions on which they want to perform an analysis.

```
## Administrative Dimensions
CREATE SPATIAL DIMENSION Municipality (
    geometry: Geometry
) WITH PROPERTIES (
    code Integer DISPLAY_STRING
);

CREATE SPATIAL DIMENSION District (
    geometry: Geometry
) WITH PROPERTIES (
    code Integer DISPLAY_STRING
) WITH PARENT (
    Municipality
);

CREATE SPATIAL DIMENSION Section (
    geometry: Geometry
) WITH PROPERTIES (
    code Integer DISPLAY_STRING
) WITH PARENT (
    District
);

## Different Spatial Dimension
CREATE SPATIAL DIMENSION Neighborhood (
    geometry: Geometry
) WITH PROPERTIES (
    code Integer DISPLAY_STRING
);

```

```

CREATE SENSOR ExampleSensor (
    ....
) WITH SPATIAL GROUP Administrative (
    Section, District, Municipality
) WITH SPATIAL GROUP NeighborhoodGroup  (
    Neighborhood
);
```

Listing 4: CREATE SPATIAL GROUPS example definition

As seen in Listing 4, three dimensions (`Section`, `District` and `Municipality`) are defined using `CREATE SPATIAL DIMENSION` and associated in a hierarchical way using the keyword `WITH PARENT`. On the other hand, another dimension `Neighborhood` is defined. When defining the sensor, two `SPATIAL GROUPS` are created, the first `Administrative` that has the first hierarchy of spatial dimensions associated, and the second group `NeighborhoodGroup` that contains the dimension `Neghborhood`. This definition of spatial groups is incorporated into the final application, allowing users to choose from two spatial groups: `NeighborhoodGroup` and `Administrative`. Enabling users to filter or aggregate sensor data more effectively, helping to conceptually distinguish between the different spatial dimensions associated with a sensor.

```
CREATE CATEGORICAL DIMENSION <dimNameRef> (
    field: <fieldName> <dataType>
);
```

Listing 5: CREATE CATEGORICAL DIMENSION sentence

As seen in Figure 6, in the same way as the user can define `Spatial Groups`, `Category Groups` can be defined; adding together same conceptual `Categorical Dimensions` associated with a sensor.

Ranges can be defined using `CREATE RANGE`, which consists of a list of ranges and a name. These ranges may consist of a single distinct value or a range denoted by the notation `x TO y`. The `COLOR` keyword can also assign a specific color to each range. Finally, the `AS` keyword will designate a label for each range.

```
CREATE RANGE <rangeNameRef> (
    propValue AS "propName1" [COLOR hexColor] ||
    FROM propValue1 TO propValue2 AS "propName1" [COLOR hexColor]
    ...
);
```

Listing 6: CREATE RANGE sentence

Finally, the statement `CREATE SENSORGROUP` (shown in Listing 7) is used to define a collection of sensors.

```
CREATE SENSORGROUP <sensorGroupRef> (
    <SensorIdRef1>,
    <SensorIdRef2>
);
```

Listing 7: CREATE SENSORGROUP sentence

The DSL parser was developed using ANTLR4[10], a parser generator that translates grammar files into code for various languages, including Java, Python, and C++. We used it to generate JavaScript code to parse our DSL definitions. Building on this, we created `sensor-dsl`, a JavaScript library[11] that functions as a standalone tool or can be integrated into a JavaScript application. It processes a DSL instance and returns the specification in JSON format.

4.2. IoT data warehouse DSL Metamodel

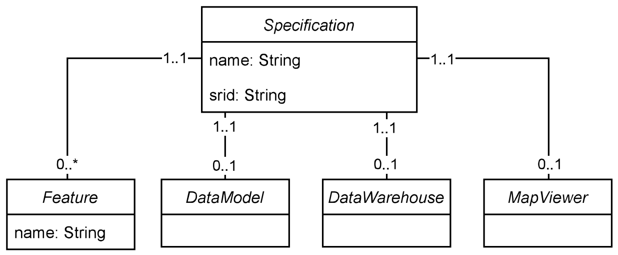


Figure 7: Metamodel of the specification that outputs the Sensor DSL Parser

The specification builder component, shown in Figure 5, receives a valid IoT data warehouse specification (i.e., and instance of the Sensor DSL metamodel shown in Figure 6) and a valid set of features selected from the SPL feature model (i.e., a list of feature names selected from the feature model that is shown in Figure 11) and builds a product specification conforming to the IoT data warehouse DSL Metamodel. This second DLS metamodel is very similar to the Sensor DSL but includes much more detail that is used for product construction in the SPL. Figure 7 shows the main structure of the IoT data warehouse DSL metamodel. A product specification includes a product name and a SRID. It is composed of set of features (taken from the feature model described in Section 4.3), a data model specification (defined in Figure 8), a data warehouse specification (described in Figure 9), and a map viewer specification (defined in Figure 10).

The data model section of the DSL (shown in Figure 8) is designed to describe both the structural and relational aspects of the data warehouse. This part of the metamodel represents a relational database model composed of a set of entities (represented by the class `Entity`). Each `Entity` comprises a set of `Field` instances representing the entity's properties. Some of the `Field` instances represent attributes of the entity, and some represent relations to other entities.

[10] https://www.antlr.org/
[11] https://github.com/lbdudc/sensor-dsl

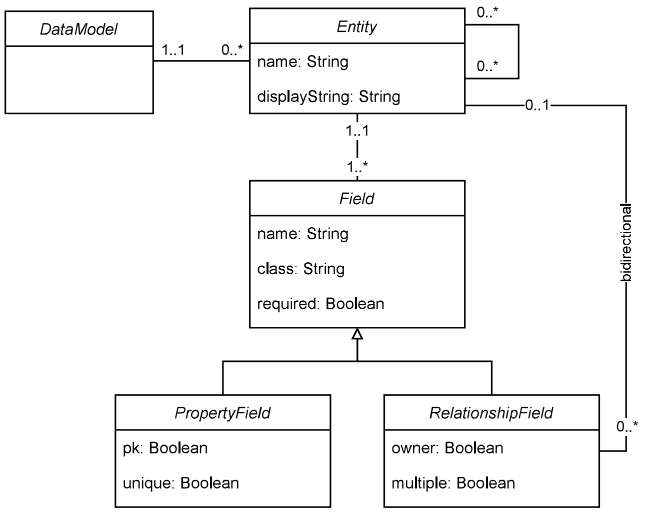


Figure 8: Metamodel DSL Parser Output: Data Model

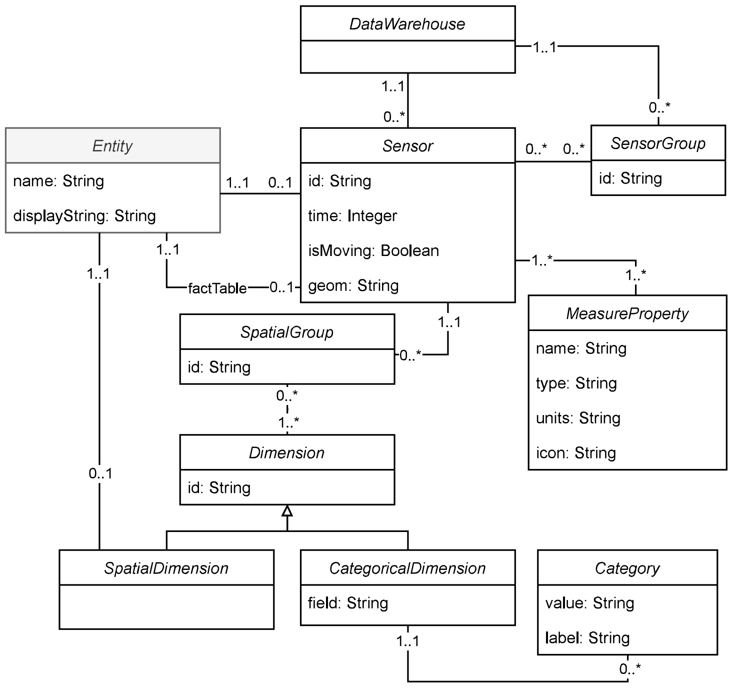


Figure 9: Metamodel DSL Parser Output: Data Warehouse

Figure 9 shows the part of the metamodel that defines the data warehouse. The information described by this metamodel is similar to that in Figure 6. The `Sensor` class respresents a sensor and it is associated with the `SensorGroup` and the `MeasureProperty`. `SpatialDimensions` and `CategoricalDimension` are unified in a specialization hierarchy to be handled simultaneously during the product derivation stage of the software product line. The most significant change with respect to the Sensor DSL of Figure 6 is the description of the storage infrastructure for the IoT data warehouse. For each sensor, two `Entity` instances are created (from Figure 8): one for the sensor's properties and another for the fact table that holds the measurements. Similarly, for each spatial dimension, an `Entity` is defined to store its information. The hierarchy of spatial dimensions is represented by the cyclical relationships between entities shown in Figure 8. This generation of the storage infrastructure allows us to represent sensor information as a data warehouse.

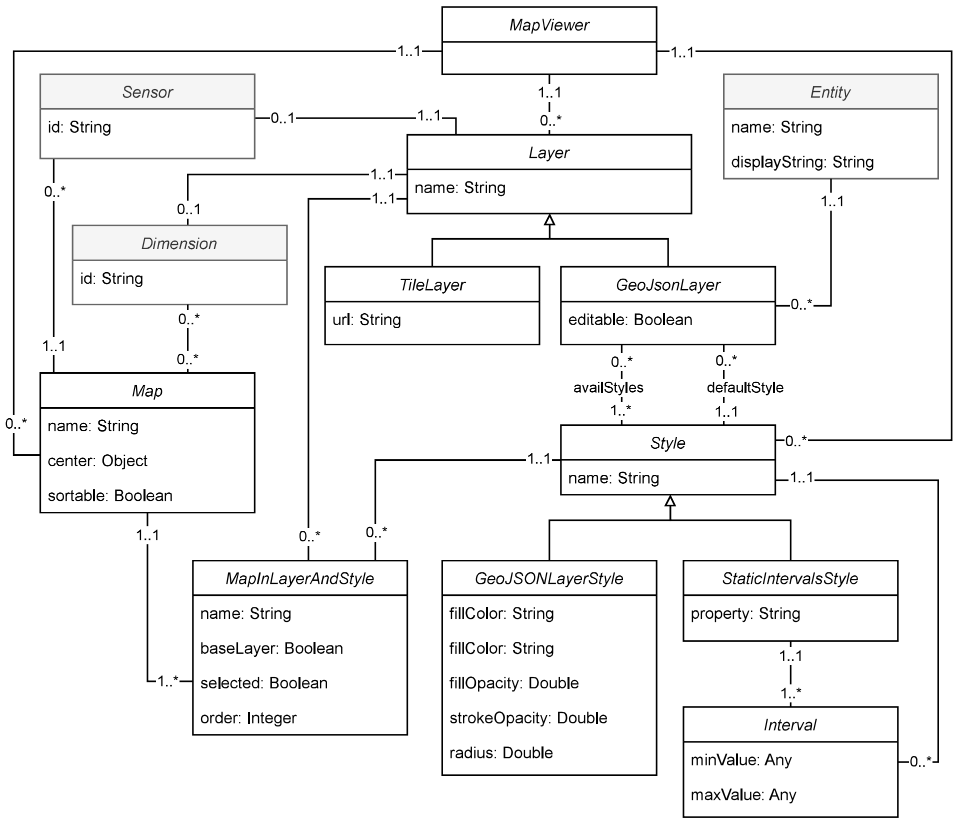


Figure 10: Metamodel DSL Parser Output: Map Viewer

Figure 10 shows the map viewer section of the metamodel. It is composed of a collection of `Layer` instances, where each layer can either be a tile-based layer used for context (such as an OpenStreetMap-based base map) or a GeoJSON layer that displays data from the warehouse (and therefore, it is associated with an `Entity`). Each layer has a collection of visualization styles (`availStyles`) and a default style (`defaultStyle`). The styles can either be a simple style, represented by the `GeoJSONLayerStyle` class, which displays all sensors with the same style, or a complex style, represented by `StaticIntervalsStyle`, which applies a different style for each interval of a range. Finally, the map viewer section includes maps displaying sensors and layers with specific styles. From the Sensor DSL instance, a `Map` is created for each defined `Sensor` to display the information. For each `Sensor`, a `GeoJSonLayer` is also created, associated with the `Entity` in which the sensor measurements are stored (i.e., the fact table). For each

Dimension, a GeoJsonLayer is created. These dimension layers are related via the MapInLayerAndStyle class to a map with a default style and associated with the sensor map where it has been defined in the DSL. For each Sensor and Dimension of any type, a GeoJSONLayerStyle is created and associated with the Layer that represents each of these classes. Furthermore, for each defined Range, a StaticIntervalStyle is created, to which a set of Interval are associated for each interval defined in the range. Each StaticIntervalStyle is associated with a GeoJSONLayer of a Dimension or an Entity, depending on whether the range was used in a categorical dimension or a property of an entity.

4.3. IoT Data warehouse Software Product Line

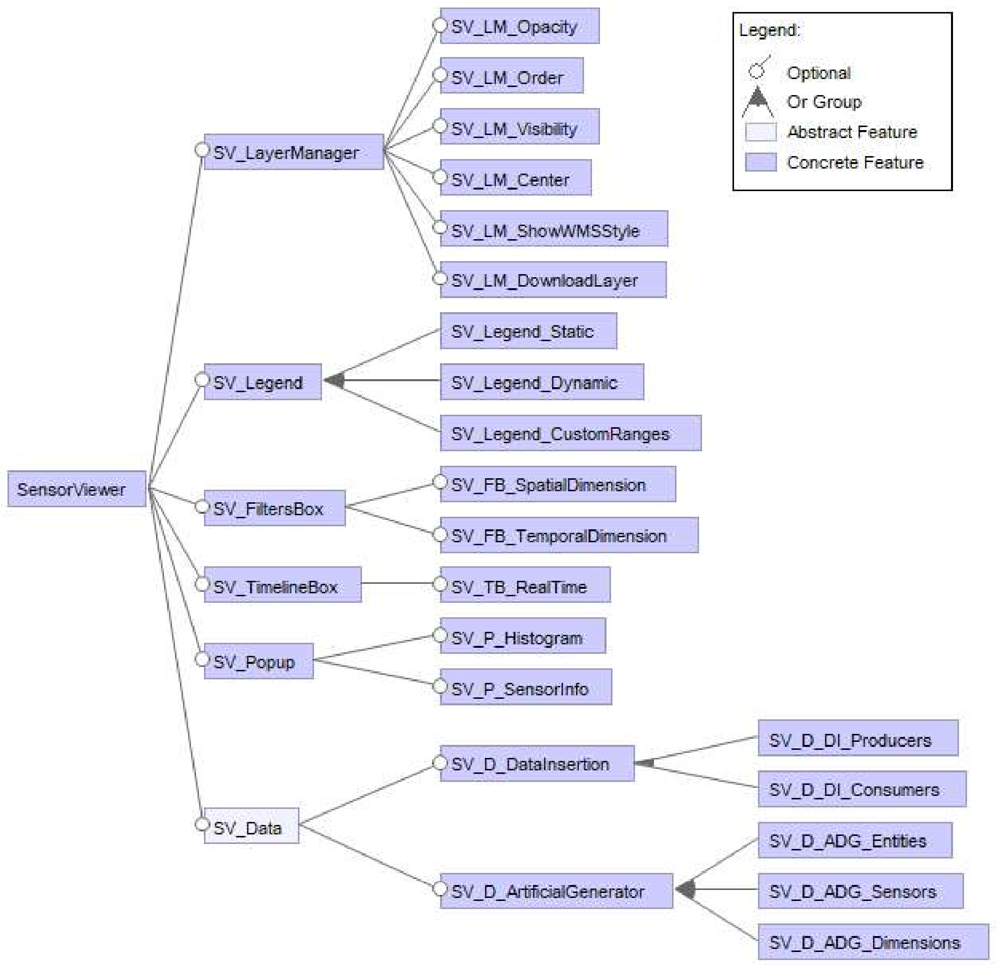


Figure 11: Feature Model of the IoT data warehouse SPL

A Software Product Line (SPL) comprises related software products with common features. This approach is designed to produce various applications efficiently within a specific context. SPLs facilitate systematic reuse while allowing for customization and variability by leveraging product similarities. We developed an SPL for IoT data warehouses with a wide range of functionalities related to sensor data, covering spatial, temporal, and categorical information. Additionally, it allows for data visualization on a map, enabling end-users to perform analyses within a web-based application. The web-based product also provides for query creation, enabling aggregations and data filtering, and workflows for data ingestion into the final system. Figure 11 shows the feature model of the SPL. A feature model represents optional, required, alternative, and mutually exclusive features in a Software Product Line (SPL), along with their constraints. It guides product configuration, helping to manage complexity, ensure consistency, and support decision-making for customized solutions (Käköla and Duenas (2006)).

The feature model defined for the IoT data warehouse focuses on enabling high-level customization of the final products. The Layer Manager (`SV_LayerManager`) allows users to manage map layers, including options to hide, reorder, and adjust their opacity. The Legend (`SV_Legend`) represents the meaning of symbols, colors, and styles associated with the map layers while enabling users to modify visualization styles by switching between fixed and dynamic ranges based on the displayed data. The Filters Box (`SV_FiltersBox`), located in the left sidebar, provides filtering capabilities based on spatial and temporal dimensions. The Timeline Box (`SV_TimelineBox`), positioned at the bottom of the interface, allows users to navigate through time and select specific instances. The Popup feature (`SV_Popup`) lets users choose geometries on the map, such as dimensions or sensors, to view detailed information about the sensors and their measurements. Lastly, the Data Ingestion feature (`SV_Data`) includes automatic data ingestion capabilities.

The architecture of the resulting products (see Figure 12) consists of a web-based front-end application developed with Vue.js. For map visualization, we use the @lbdudc/map-viewer library[12], which serves as a wrapper for the Leaflet.js library[13]. The web-based front-end manages user interactions and renders the user interface, while communication with server-side components is handled via RESTful APIs; all these interactions are handled with the @lbdudc/magical-state library[14], by getting the definition of the dimensions, sensors, their relationships, and how their possible values are retrieved, the component manages the state automatically every time the user interacts with the UI (Lamas Sardiña et al. (2023)). The back-end infrastructure is powered by a Spring Boot server, which processes business logic, handles data operations, and responds to web-based front-end requests.

In IoT applications, efficient processing of large sensor data volumes requires near-instantaneous ingestion and rapid data analysis and visualization. To manage real-time data ingestion and streaming, the products utilize Apache Kafka[15], a distributed platform for publishing, subscribing, storing, and processing records. Kafka's messaging system ensures decoupled data ingestion, fault tolerance, and scalability. The SPL-generated product includes fully developed consumer code and a producer skeleton with classes for handling sensor data and Kafka topics. End users only need

[12] `https://npmjs.com/package/@lbdudc/map-viewer`
[13] `https://leafletjs.com/`
[14] `https://github.com/lbdudc/magical-state`
[15] `https://kafka.apache.org/`

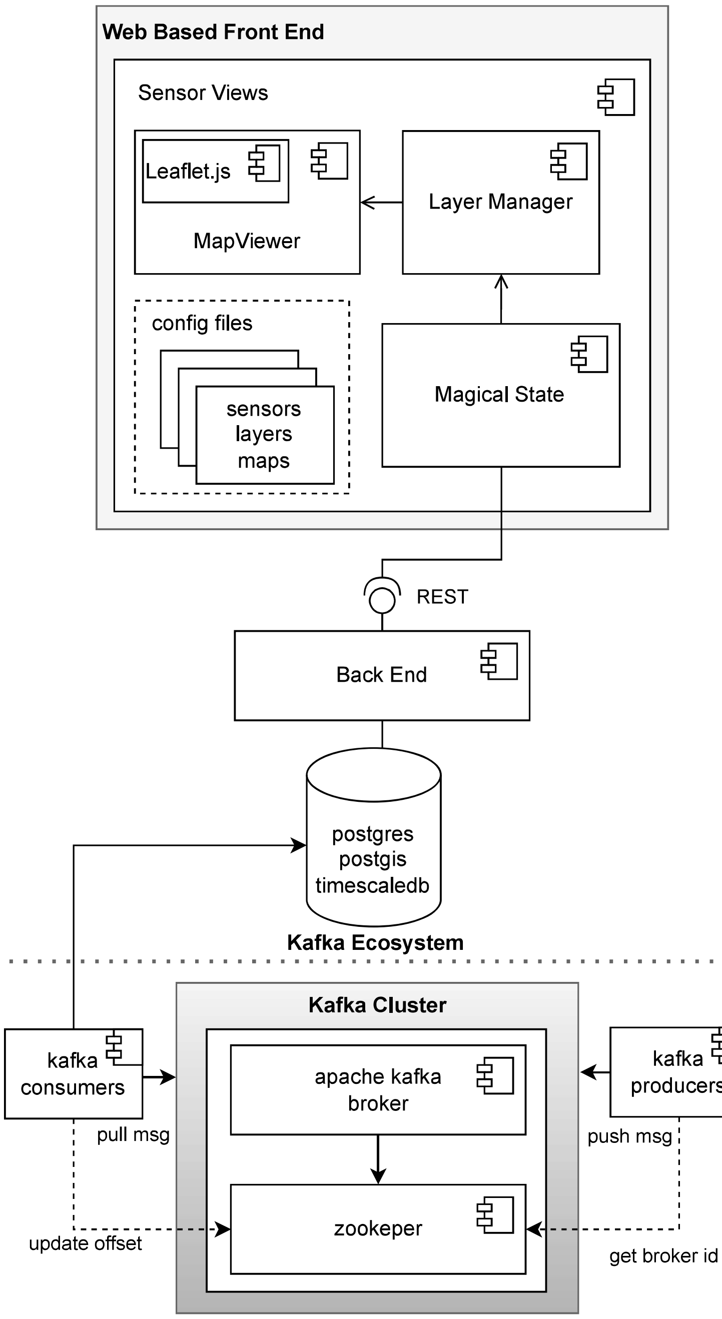


Figure 12: Architecture of the generated products by SensorPublisher

to implement logic for generating measurement data (e.g., fetching from a URL or ingesting from a CSV file).

The system uses PostgreSQL with PostGIS and TimescaleDB[16], a high-performance time-series database, to store and retrieve data. TimescaleDB uses a hybrid storage method that enables different storage mechanisms based on the data's age and usage patterns. Recent data remains row-oriented by default, ensuring high ingest rates and efficient

[16] https://timescale.com/

queries. In contrast, older data can be transformed into a highly compressed columnar format to optimize analytical queries. This architecture helps to manage query workloads and improves performance and storage efficiency. Hypertables simplify managing large datasets by automatically partitioning data by time and other dimensions. Continuous aggregations provide pre-computed summaries (e.g., daily or monthly averages) that update with new data, ensuring fast insights without repeatedly processing raw data.

For each sensor declared in the metamodel, a specific table is created to store its properties, and another table records the measurements it collects. For instance, the `MeteorologicalStation` sensor in Listing 8 would have a table for the sensor properties like `name`, `registrationDate`, and `geometry`, and another for the measurements such as `pressure` and `precipitation`. A TimeScaleDB hypertable is created for the measurements table, using the timestamp attribute for efficient partitioning. Additionally, a table stores properties and geometries for each spatial dimension defined in the DSL, such as `EmergencyZone` and `Council`. For instance, in Listing 8, two spatial dimensions are defined: `EmergencyZone` and `Council`. A table is created for each dimension to store their respective geographic objects.

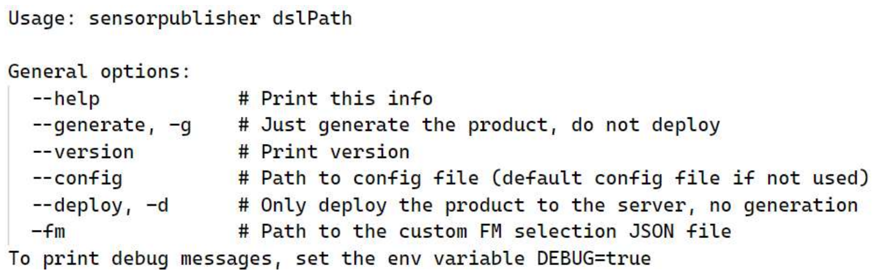
A single table stores all possible value combinations for categorical dimensions, allowing each measurement to link to a specific combination, which supports efficient data aggregation in TimeScaleDB. Furthermore, a table stores the range's start and end values for each range defined in the DSL. Database triggers generate necessary dimension value combinations (if absent) and associate the measurement with these combinations and range intervals. Since the number of possible values in each categorical dimension is usually small, creating all combinations does not result in a combinatorial explosion (this table is usually defined as a "junk dimension", a table that consists of attributes that do not belong in the fact table or any of the existing dimension tables Golfarelli et al. (1998)).

For the temporal dimension, we use TimeScaleDB's standard approach. A materialized view is defined for each aggregation interval (hour, day, week, month, or year). These aggregation intervals are determined based on the measurement interval declared for the sensor, starting with the smallest interval. For example, if the data collection interval is set to 60 seconds, the most detailed materialized view would correspond to 60 seconds, and from it, views with higher aggregation levels would be created. All of these materialized views make calculations for each property the sensor measures, making average, sums, minimum and maximum operations of values that they aggregate. These aggregation operations were chosen based on the results of creating the four cases presented below in Section 5.1. Further validation is needed for these operations to be sufficient across all potential use cases.

When mobile sensors are defined, the database undergoes some changes. The first modification is storing the measurement geometry in the measurements table rather than the sensor entity table. The second change involves the materialized views that aggregate measurements by sensors. In addition to calculating averages, maximums, and minimums, these views will aggregate the geometries. This aggregation aims to simplify the data, enabling the visualization of a series of points for a sensor's time interval. The final change is related to spatial dimensions. Triggers will be implemented to link each new measurement with its corresponding spatial dimensions. For instance, this will determine whether a measurement falls within a specific city section or a ship's measurement is within a particular ocean region. The materialized views will be duplicated to align with the same time intervals used for static sensors. However, a key difference is that these views will organize the data by spatial dimensions rather than by sensors.

To produce final custom products addressing a custom feature model selection, SPLs use a derivation engine. We created our derivation engine, a library named spl-js-engine[17]. It is a JavaScript library that, following the annotative approach, can generate final product source code from the annotated code, the feature model of the product line, and a product specification. Spl-js-engine validates the specification of the product against the feature model before generation (Cortiñas et al. (2022)).

4.4. Tool



```
Usage: sensorpublisher dslPath

General options:
  --help            # Print this info
  --generate, -g    # Just generate the product, do not deploy
  --version         # Print version
  --config          # Path to config file (default config file if not used)
  --deploy, -d      # Only deploy the product to the server, no generation
  -fm               # Path to the custom FM selection JSON file
To print debug messages, set the env variable DEBUG=true
```

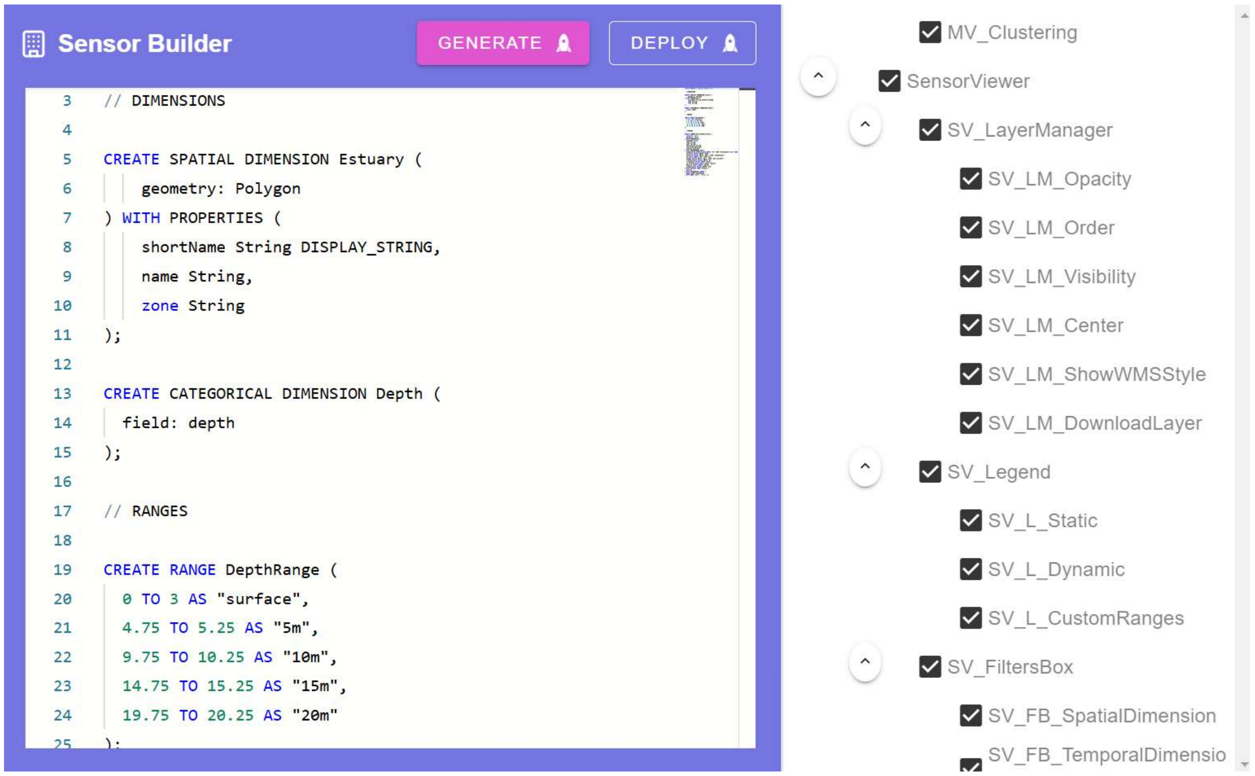
Figure 13: SensorPublisher tool command line usage

We provide two ways of interacting with SensorPublisher: a command-line interface (CLI) and a graphical user interface (GUI). Figure 13 shows the usage of the command-line interface. The user is only expected to provide the path to the DSL definition described in Section 4.1. The tool offers several options to customize the generation and deployment process. For instance, the `-fm` option specifies a JSON file containing a selection of features from the feature model, enabling flexible product customization (a default selection of features is provided if omitted).

Furthermore, users can modify the default configuration by providing a custom configuration file using the `-config` option; here also will be specified the deployment information configuration if needed. The tool will deploy the generated products to the server if the `-d` option is included. This approach provides users a powerful and versatile method to configure and manage their deployments efficiently.

The tool also provides a web-based GUI. Figure 14 shows how users can use an integrated text editor in the graphical user interface (GUI) to specify sensor characteristics. You can drag and drop text files into this editor. A graphic representation of the feature model is also offered, and users can choose desired features from this representation to customize the final product.

[17] https://github.com/AlexCortinas/spl-js-engine



```
// DIMENSIONS

CREATE SPATIAL DIMENSION Estuary (
    geometry: Polygon
) WITH PROPERTIES (
    shortName String DISPLAY_STRING,
    name String,
    zone String
);

CREATE CATEGORICAL DIMENSION Depth (
  field: depth
);

// RANGES

CREATE RANGE DepthRange (
  0 TO 3 AS "surface",
  4.75 TO 5.25 AS "5m",
  9.75 TO 10.25 AS "10m",
  14.75 TO 15.25 AS "15m",
  19.75 TO 20.25 AS "20m"
);
```

- ✓ MV_Clustering
- ✓ SensorViewer
  - ✓ SV_LayerManager
    - ✓ SV_LM_Opacity
    - ✓ SV_LM_Order
    - ✓ SV_LM_Visibility
    - ✓ SV_LM_Center
    - ✓ SV_LM_ShowWMSStyle
    - ✓ SV_LM_DownloadLayer
  - ✓ SV_Legend
    - ✓ SV_L_Static
    - ✓ SV_L_Dynamic
    - ✓ SV_L_CustomRanges
  - ✓ SV_FiltersBox
    - ✓ SV_FB_SpatialDimension
    - ✓ SV_FB_TemporalDimensio

Figure 14: SensorPublisher GUI with the DSL editor and the FM selector

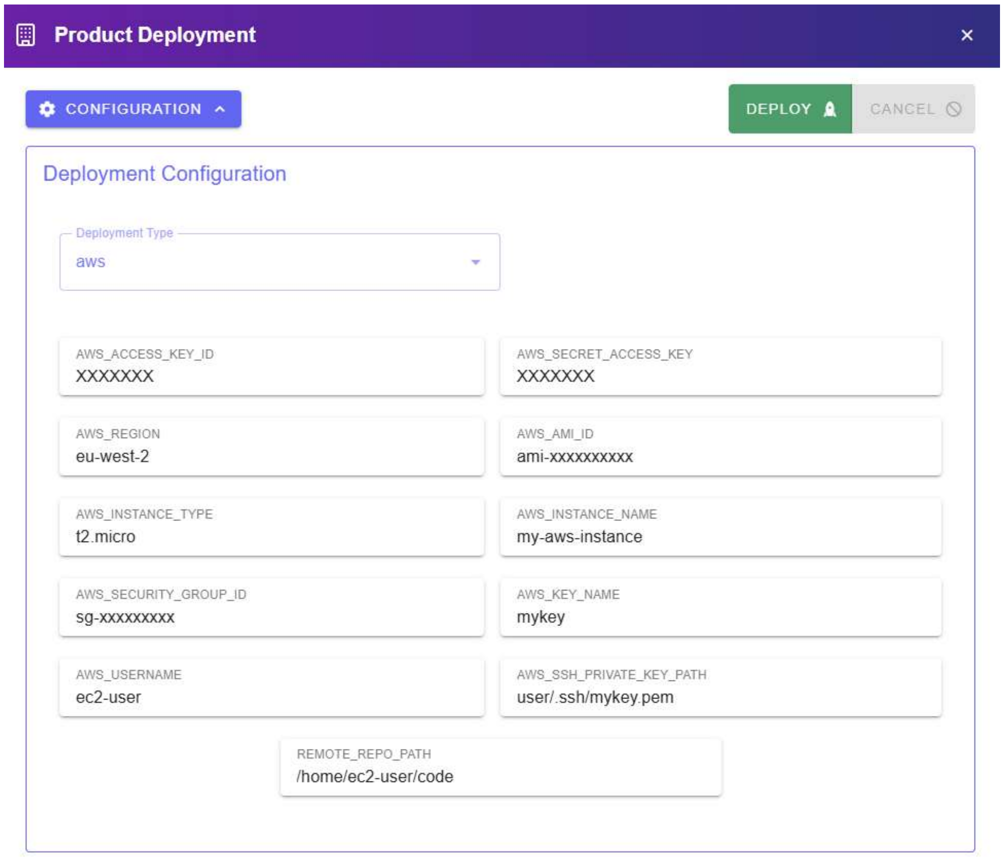
As shown in Figure 15, users can then choose to either deploy the generated product using the `Deploy` button or generate the final product using the `Generate` button. By choosing between “Local”, “AWS”, or “SSH”, the tool enables users to deploy a product. The tool will output the deployment process in real-time when the button is pressed.

5. Results and Discussion

This section uses the tool described in Section 4 to present the results of the case studies, showing how the systems implement all the functionalities outlined in Section 2. Finally, we discuss the advantages of this approach.

5.1. Results

Appendix A includes all DSL instances for the four examples: the Meteorological Monitoring System (Listing 8), the Traffic And Air Quality System (Listing 9), the Coastal Monitoring System (Listing 10) and the Oceanic Monitoring System (Listing 11). Figure 16 presents a screenshot of the resulting IoT data warehouse, highlighting various user interface aspects within designated boxes. Box 1 displays a sensor color-coded according to its measurement values based on the legend in Box 2. Box 3 contains the selector for specifying the desired level of spatial aggregation, allowing users to choose the spatial dimension and the hierarchy level for aggregation and to filter by specific geographic features. Box 4 contains the selector for defining the temporal aggregation level, while the current time is selected



Deployment Type: aws

AWS_ACCESS_KEY_ID: XXXXXXX
AWS_SECRET_ACCESS_KEY: XXXXXXX
AWS_REGION: eu-west-2
AWS_AMI_ID: ami-xxxxxxxxxx
AWS_INSTANCE_TYPE: t2.micro
AWS_INSTANCE_NAME: my-aws-instance
AWS_SECURITY_GROUP_ID: sg-xxxxxxxxx
AWS_KEY_NAME: mykey
AWS_USERNAME: ec2-user
AWS_SSH_PRIVATE_KEY_PATH: user/.ssh/mykey.pem
REMOTE_REPO_PATH: /home/ec2-user/code

Figure 15: SensorPublisher GUI Deployment configuration

using the controls at the bottom of the interface. Box 5 features a checkbox to enable real-time display (i.e., the current time). Box 6 includes the temporal selector for choosing a specific moment in time and controls for animating the visualization, which automatically advances the time. Finally, Box 7 provides the selector, allowing users to decide which of the sensor's measurements is used to display the map.

Figure 17 shows a screenshot of the IoT data warehouse for the Traffic and Air Quality System. In the user interface, aggregation by *sections* has been selected (as shown in the selector in Box 1), displaying the average value of observations for each section (as chosen in the selector in Box 2); other aggregation options are to show the maximum and the minimum values. The example demonstrates that our solution allows the definition of multiple sensors. We define two distinct sensors in our case, as shown in Listing 9. The first sensor, `TrafficSensor`, collects data on traffic intensity, average speed, and lane occupancy ratio. The second sensor, `AQSensor`, gathers information on carbon dioxide levels, particulate pollution, carbon monoxide, and other related metrics.

Figure 18 displays the Coastal Monitoring System product, showing the measurement station values on a map. The DSL defines a `BuoySensor` designed to measure water properties such as temperature, pressure, pH, oxygen,

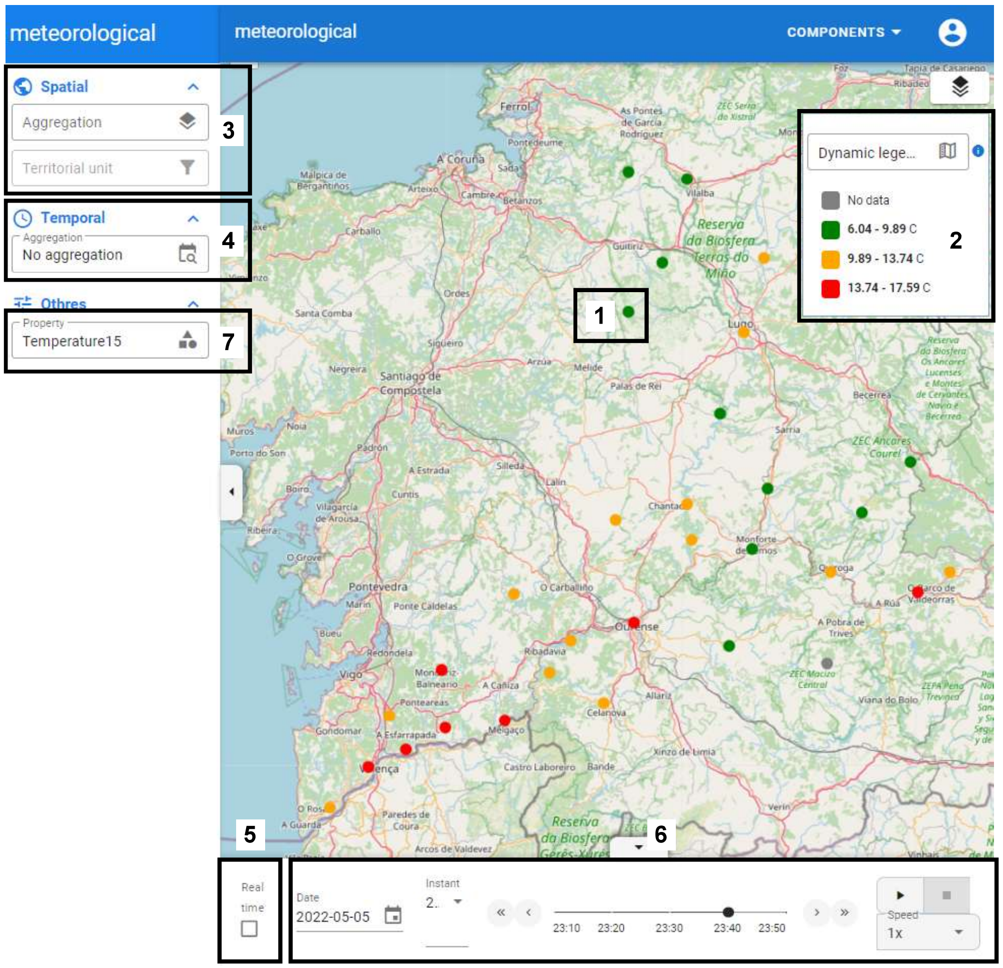


Figure 16: Generated Product for the Meteorological Monitoring System

UV fluorescence, and more. A categorical dimension for depth, with value ranges in meters, enables aggregation and filtering based on this parameter. A spatial dimension called "Estuary" has also been created to accommodate sensors from various estuaries across Galicia. Figure 19 presents a line graph illustrating the daily temperature values collected by the sensor, aggregated to highlight the variation in values based on depth.

Figure 20 shows the oceanic monitoring product. A sensor, `ShipMovingSensor`, represents ships, specifying that their movement over time is captured using the clause `CREATE MOVING SENSOR`. This sensor measures various properties, including sea pressure, salinity, and temperature. Additionally, measurements are taken based on depth, as indicated by the `CREATE CATEGORICAL DIMENSION` clause. Spatial dimensions are also created to enable aggregation by country and ocean. The system represents the ship's current position along with the corresponding measurement values at a specific time.

A time aggregation is displayed in Figure 21, which shows the various locations from which the ship has taken

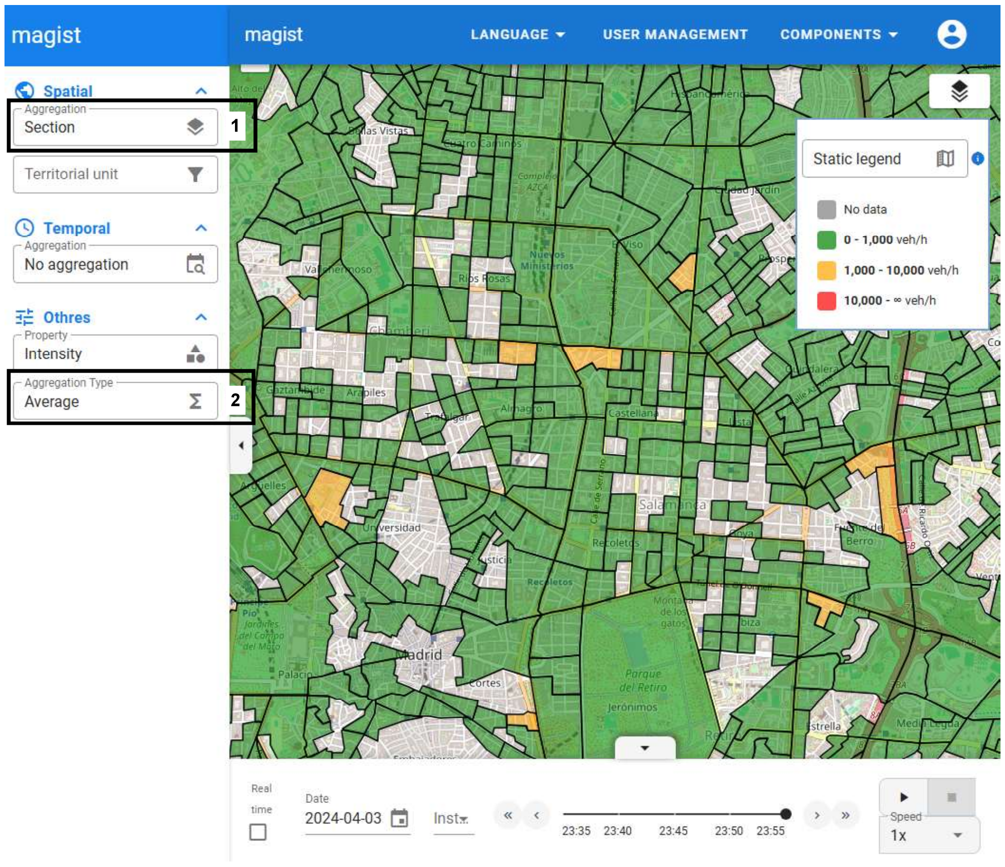
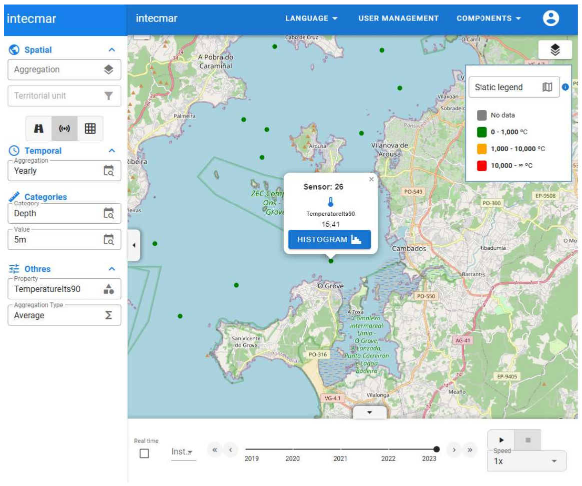


Figure 17: Generated Product for the Traffic And Air Quality System

measurements. This visualization allows us to see the colors representing each route a specific ship has traveled, with lines colored according to the measurements at particular points. Figure 22 illustrates the annual routes of all ships, each represented in a different random color.

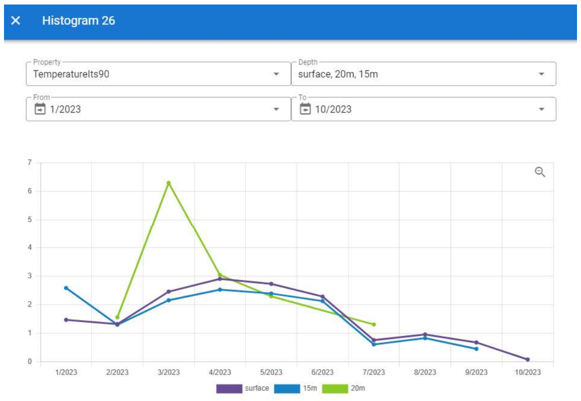
5.2. Discussion

RQ1 evaluates the feasibility of identifying common characteristics across IoT domains in geosciences and validating the creation of an SPL for an IoT data warehouse that supports these shared characteristics. The goal is to reduce development time while maintaining flexibility for domain-specific requirements. We identified common functionalities and data models and designed an approach with a DSL, feature model, and unified architecture to enable customization within the SPL. The SPL was implemented with TimescaleDB and Apache Kafka for efficient, scalable data integration. It allows product modifications while retaining SPL benefits like reduced development costs and improved reusability. The SPL's effectiveness was tested with prototype systems across IoT domains. The DSL enables defining sensors, their properties, and spatial dimensions for aggregation, with the temporal dimension automatically inferred from the measurement interval. The resulting user interface displays measurements on a map, allowing users



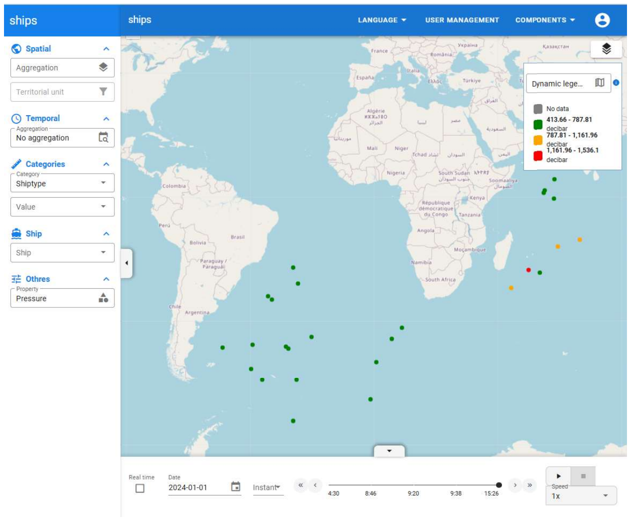
Aggregation: Yearly
Category: Depth
Value: 5m
Property: TemperatureIts90
Aggregation Type: Average
Sensor: 26
TemperatureIts90
15,41
🟩 0 - 1,000 ºC
🟧 1,000 - 10,000 ºC
🟥 10,000 - ∞ ºC
Speed: 1x

Figure 18: Generated Product for the Coastal Monitoring System: Sensor information aggregated yearly



Property: TemperatureIts90
Depth: surface, 20m, 15m
From: 1/2023
To: 10/2023

Figure 19: Line graph displaying daily measurements with average temperature values



Aggregation: No aggregation
Category: Shiptype
Property: Pressure
🟩 413.66 - 787.81 decibar
🟧 787.81 - 1,161.96 decibar
🟥 1,161.96 - 1,536.1 decibar
Date: 2024-01-01
Instant
Speed: 1x

Figure 20: Generated Product for Moving Ships case: view measurements without aggregation

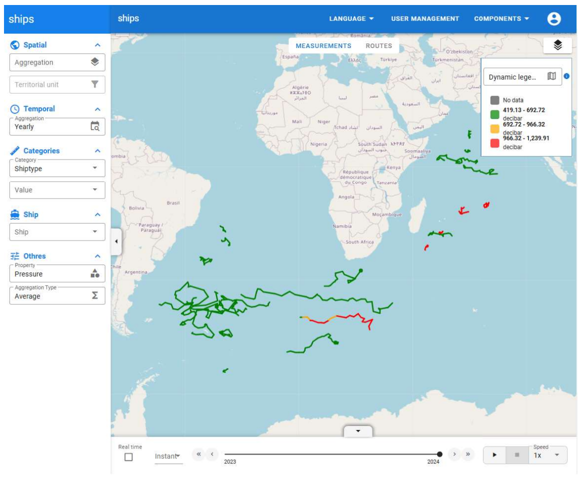


Figure 21: Generated Product for Moving Ships case: yearly aggregation showing measurements

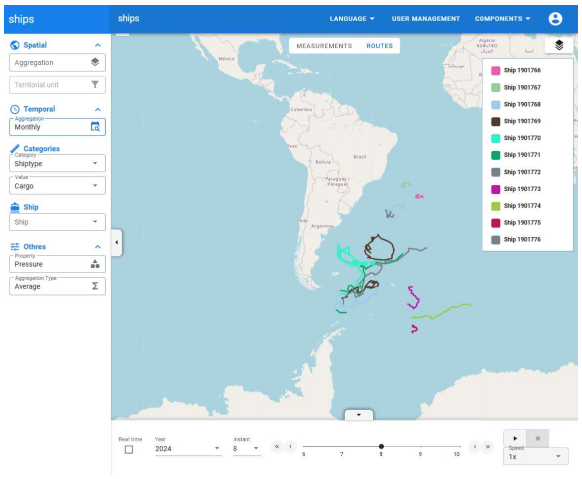


Figure 22: Generated Product for Moving Ships case: yearly aggregation showing ship routes

to select desired spatial and temporal aggregations, navigate the map to choose the spatial area, and adjust the temporal range using a control bar. These prototypes confirm that the SPL can effectively capture the commonalities across different IoT data warehouse domains. However, the applicability of the SPL to all potential domains remains unclear, offering a direction for future research to validate and expand common functionalities.

RQ2 examines the ability to describe the variability of an IoT data warehouse data model across different domains. We propose a DSL that captures this variability, supporting multiple sensor types with diverse properties, update intervals, and mobile sensor capabilities. The DSL allows flexible measurement parameters, temporal dimensions inferred from update intervals, and customizable spatial and categorical dimensions. This variability is demonstrated through four case studies. The DSL also supports diverse visualization needs by enabling the configuration of dimensions and ranges. However, limitations exist: the DSL simplifies sensor parameters, excluding complex sensors or data structures. Additionally, while the SPL produces a complete product, the responsibility for data ingestion still falls on the developer, as providing universal support for all possible alternatives in this area is challenging.

RQ3 addresses the challenge of enabling researchers to create and deploy systems that provide efficient data visualization for sensor data. We propose a low-code platform where users describe the data model using the DSL, select functionality through the feature model, and provide deployment details. The platform includes a user-friendly GUI and a command-line tool (CLI) for integration into automated pipelines. While the platform significantly lowers the technical barrier by simplifying the creation process and supporting one-click deployment, it does not entirely eliminate the need for software engineering skills. Certain tasks, such as defining sensors using the DSL or coding specific aspects of the Kafka workflow, require programming expertise. However, the platform mitigates this challenge by providing pre-made code and structured guidance. The system has been validated against real-world constraints and requirements to ensure its practical applicability. Case studies further demonstrate how it bridges technical complexity, making IoT data visualization more accessible for research, particularly in geosciences. Looking ahead, we aim to enhance accessibility by enabling model descriptions using natural language, reducing the technical knowledge required and broadening the platform's usability.

This work lays the foundation for new software tools that can further enhance the usability and automation of IoT data warehouse creation. One promising direction is the development AI-powered assistants, such as chatbots, to simplify the DSL definition process. These tools could provide real-time suggestions, autocompletion, and validation. Another natural extension would be providing an interface with graphical modeling capabilities (e.g. Martínez-Lasaca et al. (2023)). This would allow users to define data models, ingestion workflows, and visualization parameters through an intuitive visual interface instead of writing DSL code manually. Another opportunity is the creation of more advanced data ingestion tools. While the SPL provides flexibility in defining data models and visualization, future tools could facilitate data integration by offering predefined connectors for common IoT protocols and databases. Additionally, an ETL (Extract, Transform, Load) framework with a graphical interface could allow users to configure data ingestion workflows visually, making the setup process more intuitive. Further advancements could focus on deployment and optimization tools. For instance, an intelligent deployment assistant could analyze sensor configurations and recommend the best storage and processing strategies based on data volume and update frequency. Integrating the SPL with cloud-native orchestration frameworks could also enable seamless scaling and efficient resource management.

6. Conclusions

We have developed a framework for the geosciences field that addresses challenges in creating sensor-based, web-centric data warehousing systems. The first result involves creating a domain-specific language for describing sensor data warehouses, allowing researchers to define sensor types, specify properties and measurements, and customize data visualization styles. The second result is an IoT data warehouse software product line for geoscience, enabling users to filter, aggregate, and visualize data through charts like value evolution and histograms. We validated the approach in four scenarios: meteorological monitoring, traffic and air quality data collection, and seawater monitoring (both with static and moving sensors). Feedback confirms that our product line reduces the time and effort needed to implement IoT-based systems, boosting productivity and enabling focus on specific applications. The third result is a low-code tool that simplifies system development for geoscientists with limited programming skills, enabling rapid creation and deployment of web-based IoT data warehouses.

While it is challenging to demonstrate that our approach supports every possible application, the framework's functionality is sufficient for the use cases presented in this publication; however, we acknowledge that further validation is necessary to consider generalizing this approach across a broader range of use cases. In future work, we plan to evaluate the framework in broader contexts. We plan to improve the final stage of development, where users input sensor data and measurements. Currently, this requires implementing Apache Kafka producers or directly inserting data into the database. We aim to automate this process or provide user-friendly tools for easier data ingestion. Additionally, we seek to enhance the end-user experience during the iterative process of defining and testing systems with the DSL by offering the option to generate artificial sensor data, allowing users to visualize the product's output before real data is input. Finally, we aim to formally validate the DSL with field experts to ensure it covers all cases. We also plan to explore the potential of large language models to enhance the DSL's usability, enabling users to define configurations using natural language without needing to learn the grammar or consult extensive documentation.

Acknowledgments

CITIC is funded by the Xunta de Galicia through the collaboration agreement between the Department of Culture, Education, Vocational Training and Universities and the Galician universities for the reinforcement of the research centers of the Galician University System (CIGUS); partially funded by MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR: [PLAGEMIS: TED2021-129245B-C21]; partially funded by MCIN/AEI/ 10.13039/501100011033 and EU/ERDF A way of making Europe: [EarthDL: PID2022-141027NB-C21]; partially funded by Galicia Marine Science programme, which is part of the Complementary Science Plans for Marine Science of Ministerio de Ciencia, Innovación Universidades included in the Recovery, Transformation and Resilience Plan (PRTR-C17.I1), funded through Xunta de Galicia with NextGenerationEU and the European Maritime Fisheries and Aquaculture Funds.

Code availability section

The main SensorPublisher repository contains all implementation details and source code and is available for public access on GitHub at `https://github.com/lbdudc/sensor-publisher`. The needed annotated code for the SPL is published into another different repository in `https://github.com/lbdudc/sensor-publisher-module`. The repository includes all the necessary files, setup instructions, and documentation to replicate the experiments described in this study. The code is provided under an open-source license (MIT).

Repository Name: SensorPublisher

Current Code Version: v1.0.1

Contact: victor.lamas@udc.es

Program language: Javascript, VueJs, Nodejs

License: MIT

Code Versioning System: Git

Program size: 158 KB

Appendix

A. DSL example definitions

```
CREATE PRODUCT meteorologicalExample USING 4326;

CREATE SPATIAL DIMENSION EmergencyZone (
    geometry: Polygon
) WITH PROPERTIES (
    name String DISPLAY_STRING
);

CREATE SPATIAL DIMENSION Council (
    geometry: Polygon
) WITH PROPERTIES (
    name String DISPLAY_STRING
) WITH PARENT (
    EmergencyZone
);

CREATE SENSOR MeteorologicalStation (
    interval: 500,
    geometry: Point
) WITH PROPERTIES (
    name String,
    registrationDate DateTime
) WITH MEASUREMENT DATA (
    pressure Double UNITS "hPa",
    precipitation Double UNITS "L/m2",
    relativeHumidity Double UNITS "%",
    solarRadiation Double UNITS "W/m2",
    temperature90 Double UNITS "C",
    windSpeed Double UNITS "m/s"
) WITH SPATIAL GROUP Administrative (
    EmergencyZone, Council
) WITH BBOX ([50.8, -10.1], 9);
```

Listing 8: Case Study DSL definition - Meteorological Status in Galicia Province

```
CREATE PRODUCT  trafficAQExample USING SRID 4326;

CREATE SPATIAL DIMENSION Municipality (
    geometry: Geometry
) WITH PROPERTIES (
    code Integer DISPLAY_STRING
);

CREATE SPATIAL DIMENSION District (
    geometry: Geometry
) WITH PROPERTIES (
    code Integer DISPLAY_STRING
) WITH PARENT (
    Municipality
);

CREATE SPATIAL DIMENSION Section (
    geometry: Geometry
) WITH PROPERTIES (
    code Integer DISPLAY_STRING
) WITH PARENT (
    District
);

CREATE SPATIAL DIMENSION Neighborhood (
    geometry: Geometry
) WITH PROPERTIES (
    name String DISPLAY_STRING
);

CREATE RANGE IntensityRange (
    0 TO 100 AS "very-low" COLOR "#55eb34",
    100 TO 500 AS "low" COLOR "#3ba324",
    500 TO 1000 AS "medium" COLOR "#fcf11e",
    ...
);

CREATE SENSOR TrafficSensor (
    interval: 350,
    geometry: Point
) WITH PROPERTIES (
    description String,
   name String
) WITH MEASUREMENT DATA (
   intensity Double UNITS "veh/h" RANGE IntensityRange,
   ocupation Double UNITS "%",
   speed Double UNITS "km/h" ICON "speed",
) WITH SPATIAL GROUP Administrative (
   Section
) WITH SPATIAL GROUP Neighborhood (
   Neighborhood
) WITH BBOX ([45.50, -4.8], 11);

CREATE SENSOR AQSensor (
   interval: 3600,
   geometry: Point
) WITH PROPERTIES (
   address String
) WITH MEASUREMENT DATA (
   so2 Double,
   no2 Double,
   co Double,
   o3 Double,
   btx Double,
   pm10 Double,
   pm25 Double
) WITH SPATIAL GROUP Administrative (
   Section
) WITH SPATIAL GROUP Neighborhood (
   Neighborhood
) WITH BBOX ([60.50, -4.7], 10);
```

Listing 9: Case Study DSL definition - Traffic and Quality Air in Madrid

```
CREATE PRODUCT intecmar USING 4326;

CREATE SPATIAL DIMENSION Estuary (
   geometry: Polygon
) WITH PROPERTIES (
   name String DISPLAY_STRING
);

CREATE CATEGORICAL DIMENSION Depth (
    field: depth Double
);

CREATE RANGE DepthRange (
    0 TO 4.74 AS "surface",
    4.75 TO 5.24 AS "5m",
    5.25 TO 10.24 AS "10m",
    10.25 TO 15.24 AS "15m",
    15.25 TO 20.24 AS "20m"
);

CREATE SENSOR BuoySensor (
    interval: 300,
    geometry: Point
) WITH PROPERTIES (
    description String,
    maxDepth Float,
    name String
) WITH MEASUREMENT DATA (
    density Double UNITS "kg/m3",
    irradiance Double UNITS "W/m2",
    oxygen Double UNITS "mg/l" ICON "gas-cylinder",
    ph Double UNITS "pH" ICON "ph",
    pressure Double UNITS "dbar",
    salinity Double UNITS "PSU",
    temperatureITS90 Double UNITS "C" ICON "thermometer",
    transmittance Double UNITS "m",
    uv_flourescence Double UNITS "mg/m3"
) WITH SPATIAL GROUP Estuary (
    Estuary
) WITH CATEGORICAL GROUP Depth (
    Depth RANGE DepthRange
) WITH BBOX ([50.5, -9.0], 10);
```

Listing 10: Case Study DSL definition - Marine Area Monitoring System

```
CREATE PRODUCT movingShipsExample USING 4326;

CREATE SPATIAL DIMENSION Country (
    geometry: Polygon
) WITH PROPERTIES (
    name String DISPLAY_STRING
);

CREATE SPATIAL DIMENSION Ocean (
    geometry: Polygon
) WITH PROPERTIES (
    name String DISPLAY_STRING
);

CREATE CATEGORICAL DIMENSION ShipType (
    field: shiptype String
);

CREATE CATEGORICAL DIMENSION Depth (
    field: depth Double
);

CREATE RANGE DepthRange (
    0 TO 4.74 AS "surface",
    4.75 TO 5.24 AS "5m",
    5.25 TO 10.24 AS "10m",
    10.25 TO 15.24 AS "15m",
    15.25 TO 20.24 AS "20m"
);

CREATE MOVING SENSOR ShipMovingSensor (
  interval: 300,
  geometry: Point
) WITH PROPERTIES (
  description String,
  name String
) WITH MEASUREMENT DATA (
  pressure Double UNITS "dB",
  salinity Double UNITS "psu",
  temperature Double UNITS "degree_celsius"
) WITH SPATIAL GROUP Country_Ocean (
  Country, Ocean
) WITH CATEGORICAL GROUP (
  ShipType, Depth RANGE DepthRange
```

```
);
```

Listing 11: Case Study DSL definition - Ships Monitoring System

References

Alvarado, S.H., Cortiñas, A., Luaces, M.R., Pedreira, O., Places, A.S., 2020. Developing web-based geographic information systems with a dsl: Proposal and case study. Journal of Web Engineering 19, 167–194. URL: https://journals.riverpublishers.com/index.php/JWE/article/view/3351, doi:10.13052/jwe1540-9589.1923.

Basciani, F., Rossi, M.T., De Sanctis, M., et al., 2020. Supporting smart cities modeling with graphical and textual editors, in: {STAF} 2020 Workshop Proceedings: 4th Workshop on Model-Driven Engineeringfor the Internet-of-Things, 1st International Workshop on ModelingSmart Cities, and 5th International Workshop on Open and OriginalProblems in Software Language Engineering co-located with SoftwareTechnologies: Applications and Foundations federation of conferences {(STAF} 2020), Bergen, Norway, June 22-26, 2020, CEUR-WS. org. pp. 9–19.

Bourgeois, K., Robert, S., Limet, S., Essayan, V., 2018. Geoskelsl: A python high-level dsl for parallel computing in geosciences. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 10862 LNCS, 839 – 845. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85049040750&doi=10.1007%2f978-3-319-93713-7_83&partnerID=40&md5=08924ed272b77d0de425725a5f343e95, doi:10.1007/978-3-319-93713-7_83. cited by: 0.

Chan, E., Ueda, K., 2000. Efficient query result retrieval over the web, in: Proceedings Seventh International Conference on Parallel and Distributed Systems (Cat. No.PR00568), pp. 161–170. doi:10.1109/ICPADS.2000.857695.

Chaudhary, H.A.A., Guevara, I., John, J., Singh, A., Margaria, T., Pesch, D., 2022. Low-code internet of things application development for edge analytics, in: IFIP International Internet of Things Conference, Springer. pp. 293–312.

Chaves, A.J., Martín, C., Llopis Torres, L., Díaz, M., Fernández-Ortega, J., Barberá, J.A., Andreo, B., 2025. A soft sensor open-source methodology for inexpensive monitoring of water quality: A case study of no3- concentrations. Journal of Computational Science 85. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85215358863&doi=10.1016%2fj.jocs.2024.102522&partnerID=40&md5=d3f98e04f2d0ac27b77815a377c3ab33, doi:10.1016/j.jocs.2024.102522. cited by: 0; All Open Access, Hybrid Gold Open Access.

Chen, Y., Hayawi, K., He, J., Song, H., Wang, J., 2023. Impact and challenges of intelligent iot in meteorological science. IEEE Internet of Things Magazine 6, 58–63.

Cortiñas, A., Luaces, M.R., Pedreira, O., 2022. spl-js-engine: a javascript tool to implement software product lines, in: Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume B, Association for Computing Machinery, New York, NY, USA. p. 66–69. URL: https://doi.org/10.1145/3503229.3547035, doi:10.1145/3503229.3547035.

Cortiñas, A., Lamas Sardiña, V., Luaces, M.R., 2024. Sensorpublisher: Applying software product lines to the development of iot dashboards, in: Proceedings of the 28th ACM International Systems and Software Product Line Conference (SPLC 2024), Luxemburgo. pp. 153–163.

Erazo-Garzon, L., Quinde, K., Bermeo, A., Cedillo, P., 2023. A domain-specific language and model-based engine for implementing iot dashboard web applications, in: Conference on Information and Communication Technologies of Ecuador, Springer. pp. 412–428.

Erraissi, A., Belangour, A., 2018. Data sources and ingestion big data layers: meta-modeling of key concepts and features. International Journal of Engineering & Technology 7, 3607–3612.

Fowler, M., 2010. Domain-specific languages. Pearson Education.

Golfarelli, M., Maio, D., Rizzi, S., 1998. The dimensional fact model: A conceptual model for data warehouses. International Journal of Cooperative Information Systems 7, 215–247.

Käköla, T., Duenas, J.C., 2006. Software product lines. Springer.

Kefalakis, N., Roukounaki, A., Soldatos, J., 2019. Configurable distributed data management for the internet of the things. Information 10, 360.

Lamas Sardiña, V., Cortiñas, A., Luaces, M.R., Pedreira, O., 2023. Component for the visualization of a spatio-temporal data warehouse in a geographic information system. Kalpa Publications in Computing , 139–141.

Lamas Sardiña, V., De Castro Celard, D., Cortiñas, A., Luaces, M.R., 2024. Gis-publisher: Simplifying web-based gis application development for enhanced data dissemination. SoftwareX .

Ma, X., 2017. Linked geoscience data in practice: where w3c standards meet domain knowledge, data visualization and ogc standards. Earth Science Informatics 10, 429 – 441. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85019938692&doi=10.1007%2fs12145-017-0304-8&partnerID=40&md5=e8065799b2d04f5fe4f1cd47a719bfa6, doi:10.1007/s12145-017-0304-8. cited by: 28; All Open Access, Green Open Access.

Mardani Korani, Z., Moin, A., Rodrigues da Silva, A., Ferreira, J.C., 2023. Model-driven engineering techniques and tools for machine learning-enabled iot applications: A scoping review. Sensors 23, 1458.

Martinez, E., Pfister, L., Stauch, F., 2024. Developing a novel application to digitalize and optimize construction operations using low-code technology, p. 283 – 290. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85199592655&doi=10.22260%2fISARC2024%2f0038&partnerID=40&md5=8a57dd5971d1bfd72640a2dfbf401a17, doi:10.22260/ISARC2024/0038. cited by: 0.

Martínez-Lasaca, F., Díez, P., Guerra, E., de Lara, J., 2023. Dandelion: A scalable, cloud-based graphical language workbench for industrial low-code development. Journal of Computer Languages 76, 101217. URL: https://www.sciencedirect.com/science/article/pii/S2590118423000278, doi:https://doi.org/10.1016/j.cola.2023.101217.

Ortiz, J.A., Zamudio-García, V.M., 2025. Optimising water use through smart models and artificial intelligence. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-86000257775&doi=10.4018%2f979-8-3693-8074-1.ch008&partnerID=40&md5=44c696e268a7221a0c21ffd81bcbda28, doi:10.4018/979-8-3693-8074-1.ch008. cited by: 0.

Pham, L.M., Tchana, A., Donsez, D., Zurczak, V., Gibello, P.Y., De Palma, N., 2015. An adaptable framework to deploy complex applications onto multi-cloud platforms, in: The 2015 IEEE RIVF International Conference on Computing & Communication Technologies-Research, Innovation, and Vision for Future (RIVF), IEEE. pp. 169–174.

Raj, D., Prabha, P.A., Pai, A.S., Hridul, N., 2024. A comprehensive survey on iot ocean observatory systems, in: 2024 10th International Conference on Communication and Signal Processing (ICCSP), IEEE. pp. 130–135.

Rojas, E., Bastidas, V., Cabrera, C., 2020. Cities-board: a framework to automate the development of smart cities dashboards. IEEE Internet of Things Journal 7, 10128–10136.

Silva, J., Lopes, M., Avelino, G., Neto, P.S., 2023. Low-code and no-code technologies adoption: A gray literature review, p. 388 – 395. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85165946436&doi=10.1145%2f3592813.3592929&partnerID=40&md5=cb59021284d42a5b51b1cd42623b4a7e, doi:10.1145/3592813.3592929. cited by: 5.

Sreyas, N., Venkatesh, T., Harjit, R., Prabha, B., 2025. Iot integrated air quality detection and alert system. Communications in Computer and Information Science 2362, 3 – 13. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85219191123&doi=10.1007%2f978-3-031-82386-2_1&partnerID=40&md5=102f516195964dab79bb588f2ff864ee, doi:10.1007/978-3-031-82386-2_1. cited by: 0.

Vögler, M., Schleicher, J.M., Inzinger, C., Dustdar, S., 2017. Ahab: A cloud-based distributed big data analytics framework for the internet of things. Software: Practice and Experience 47, 443–454.

Yang, J., Yu, M., Liu, Q., Li, Y., Duffy, D.Q., Yang, C., 2022. A high spatiotemporal resolution framework for urban temperature prediction using iot data. Computers & Geosciences 159, 104991.