SensorPublisher: Applying Software Product Lines to the Development of IoT Dashboards

Alejandro Cortiñas*
Universidade da Coruña, CITIC
Research Center, Database Lab.
A Coruña, Spain
alejandro.cortinas@udc.es

Victor Lamas
Universidade da Coruña, CITIC
Research Center, Database Lab.
A Coruña, Spain
victor.lamas@udc.es

Miguel R. Luaces
Universidade da Coruña, CITIC
Research Center, Database Lab.
A Coruña, Spain
miguel.luaces@udc.es

ABSTRACT

Geosciences have witnessed a revolution in data collection thanks to the Internet of Things (IoT), which has made it possible to monitor complex phenomena using sensor networks. However, developing web-centric, sensor-based, data warehousing information systems presents challenges because of their complexity and cost. This paper presents an intuitive low-code development system (called SensorPublisher), based on a software product line (SPL) and a domain-specific language (DSL), that speeds up the creation of data warehousing applications for geographic sensor data. SensorPublisher allows the geoscientist to define the sensor network, to generate a software product, and to deploy the product to a local or a remote server. Our tool seeks to encourage scientists to share the outcomes of their sensor data analysis projects with their communities by means of a simple, user-friendly and cost-effective approach. We showcase the system in different geoscientific domains, such as meteorological monitoring services, traffic data and air quality monitoring in urban areas, and marine area monitoring systems.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; **Reusability**; *Abstraction, Modeling and Modularity*; *Software architectures.*

KEYWORDS

Internet of Things (IoT), Software Product Line (SPL), Domain Specific Language (DSL)

ACM Reference Format:
Alejandro Cortiñas, Victor Lamas, and Miguel R. Luaces. 2024. SensorPublisher: Applying Software Product Lines to the Development of IoT Dashboards. In *28th ACM International Systems and Software Product Line Conference (SPLC '24), September 2–6, 2024, Dommeldange, Luxembourg*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3646548.3672582

*All authors have contributed equally and the names are listed in alphabetical order.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SPLC'24, September 2-6, 2024, Luxembourg
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/10.1145/3646548.3672582

1 INTRODUCTION

The Internet of Things (IoT) has revolutionized data collecting in several disciplines and brought about a paradigm change in modern geosciences. Researchers looking to gain a thorough understanding of Earth's dynamic processes are integrating an ever-growing variety of sensor data. Oceanography, mobility studies, and environmental sciences are just a few of the areas that employ systems that combine IoT and geographic data. By employing specialized sensors, sensor networks, and real-time data collection, these technologies offer valuable insights into complex phenomena like monitoring oceanographic parameters, traffic patterns, air quality evaluation, or meteorological monitoring.

Developing IoT dashboards that exploit the data collected is a significant challenge for geoscience researchers. It involves managing complex data formats, understanding advanced information systems technology, and developing functionality for data visualization and interpretation. Data exploration, information filtering, and aggregation using time, space, attribute values, visually appealing charts, and histograms are all functionalities that researchers want to have at their disposal. However, this kind of system is complicated to develop and needs advanced skills in many areas, such as:

- Ingestion and reception of real-time sensor data that needs the use of a stream processing module.
- Storing the data requires knowledge of how to create and manage database systems.
- A backend must be created in which queries can be made to the database with information on both sensors and the measurements they record. The system must be able to make aggregations and apply filters of millions of sensor measurements, these aggregations being both spatial (e.g., predefined zones by the user, administrative zones) or temporal (e.g., hourly, daily, weekly, etc), and do so while obtaining response times that are not very high for a good end-user experience.
- A web application must be developed, allowing final users to view the results from aggregating and filtering sensors, representing them in an interactive map. It must also allow the exploration of the values evolution for each of the sensors or the spatial aggregations.

As a consequence, geoscience researchers frequently discover that their productivity and efficiency are limited by this complexity because it is not just about creating the system and its features; it is also crucial to ensure scalability and optimize response times.

Although each IoT dashboard is different from all the others, they all share several common features such as i) using a stream-oriented

data ingestion mechanism to collect the data from the sensors, ii) being based on a data lake because of the heterogeneity of the data, iii) providing tools for discovery and exploration of the information to determine if the data sets are relevant and appropriate, iv) offering a set of processing tools to work with the data in the data lake and generate new information, v) and interacting with the system using a geographic information system-based tool to visualize the information and results on a map, in addition to traditional listings. Despite the variations that exist across systems, the presence of shared features suggests that they can be managed as a family of software products, potentially utilizing variability management techniques.

In this paper, we describe a framework that creates web-centric, sensor-based, data warehousing information systems. Our contributions center around three main areas:

- **A software product line**: We describe a software product line designed specifically for information systems that collect and visualize data from sensor networks.
- **A domain-specific language**: We propose a DSL for defining and working with sensor measurements and their properties. This DSL is a useful tool for domain experts who not familiar with code development.
- **A low-code development tool**: We propose a tool that can be used via command line or using a web browser to create and deploy complex sensor-based information systems.

The rest of the paper is structured as follows. An analysis of previous work in the areas of software product lines and domain-specific languages related to geographic information systems is presented in Section 2 because it is used as a foundation for this paper. The suggested framework is described in Section 3, along with details about the DSL definition, tool architecture and usage, insights into the Software Product Line, and architecture of the generated products. In Section 4 we present experimental results on three scenarios in the geosciences field using the framework. In Section 5 the paper concludes with a summary of the results obtained.

2 BACKGROUND

2.1 Creating web-based GIS using Software Product Lines and Domain Specific Languages

SPLs have changed software development by enabling the creation of diverse products through systematic artifact reuse. Various strategies and tools have emerged to create and manage SPLs tailored to address specific challenges in different domains. Our contribution to this field is SPL-JS-Engine [4], which is a JavaScript library that utilizes an annotative approach to generate final product source code from feature models, annotated code, and product specifications. We used SPL-JS-Engine to develop an SPL specifically tailored for Geographic Information Systems (GIS) [5].

However, we found it difficult to express the variability of the products using only a feature model. Researchers have investigated how to combine product line engineering and domain-specific languages (DSL) [7] while maintaining the separation of problem and solution domains [9]. Integrating product-line architectures and DSLs is effective in managing feature models at scale [1] and developing command-and-control simulators [3]. With these advantages in mind, we presented a DSL [2] created exclusively for web-based GIS application development. This DSL gives developers the ability to define the application visualization model (i.e., maps, layers, styles, and entity specifications) in a declarative language.

Finally, we continued our research to improve SPL techniques for GIS creation and released GIS-Publisher [6], an SPL designed for web-based GIS production, which was released together with a specific DSL. This toolkit automates repetitive procedures required in product development and deployment, while also streamlining the publication and exchange of geographic data.

2.2 OGC SensorThings

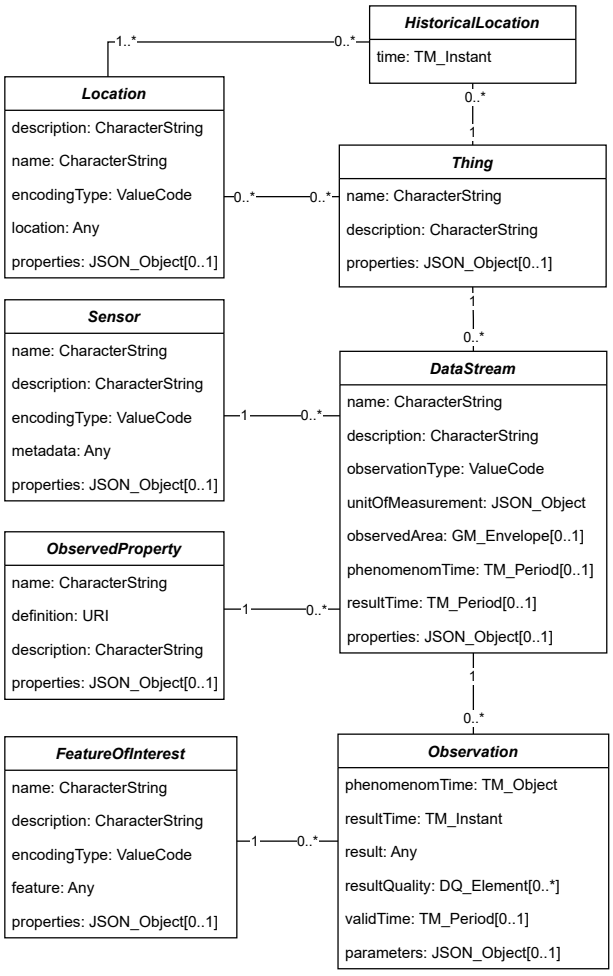

Figure 1: OGC SensorThings Model [8]

One of the top priorities for the Open Geospatial Consortium (OGC[1]) has been the development of standards that solve interoperability problems. Over the course of the past few years, the OGC SensorThings API footnotehttps://www.ogc.org/standard/

[1]https://www.ogc.org/

sensorthings/ has gained a growing amount of popularity. The SensorThings Standard provides a simplified and standardized method for defining, retrieving, and managing sensor systems that are part of the IoT. Given its relevance in geosciences, we use the SensorThings Standard in our IoT Dashboard to ensure interoperability. Specifically, we align our concepts with key SensorThings entities such as `Thing`, `DataStream`, and `Observation`. A `Thing` refers to a physical or virtual entity that can be sensed. A `DataStream` represents a series of observations or measurements collected over time generated by a `Thing`. An `Observation` represents a single measurement or reading captured by a sensor at a specific point in time.

3 A FRAMEWORK FOR CREATING IOT DASHBOARDS

Our framework consists of an automated workflow that helps geoscientists to remain focused on the domain problem, rather than being distracted by the details of the final product code. We show the high-level workflow in Figure 2. The steps of the workflow are the following:

(1) **Product Description:** Users begin by defining the sensor data model using a DSL. They also specify desired product features and deployment preferences.
(2) **Sensor Publisher:** The core component in the framework orchestrates the entire workflow. It checks that the user-provided sensor definition is correct and confirms that the feature selection is a legitimate configuration given the restrictions of the feature model. Once validated, it generates a complete product specification containing deployment details, feature model selections, and sensor definitions.
(3) **Sensor Builder SPL:** This component uses the information from the specification to generate source code based on the software product line annotated code, fulfilling all user specifications.
(4) **Product Deployer:** This component handles product deployment, currently providing three options: local machine deployment, SSH deployment (Ubuntu/Debian), and AWS deployment.

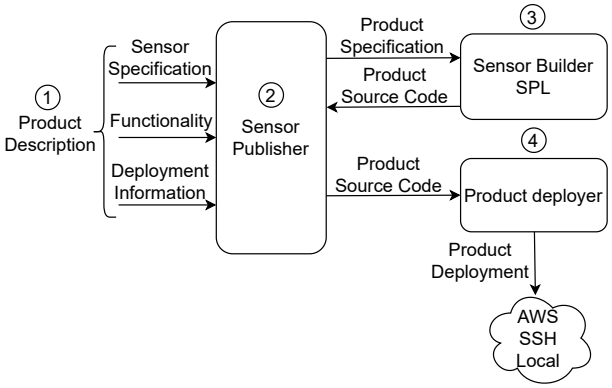

Figure 2: Framework Workflow

The following sections detail each step of the workflow, providing explanations of the process at each stage.

3.1 Product Description

As depicted in Figure 2, users initially define the specification of the sensors, followed by customizing the software product through the selection of features from the provided Feature Model, and they finally specify the deployment information for the final product. We will now describe the way they provide this information.

3.1.1 ***Domain Specific Language for Sensors Definition.*** We decided to create our domain-specific language to describe the sensor network and the IoT dashboard by building upon the conceptual model defined by the OGC SensorThings API standard. We wanted the metamodel of our language to extend SensorThings to ensure interoperability. The metamodel is shown in Figure 3.

`Sensor` and `Measurement` are the core classes of our metamodel. The class `Sensor` represents the collection of sensors of a given type and thus it is a specialization of the `DataStream` class from the OGC SensorThings API standard. Each sensor type has a name, an interval that defines how often the sensor will collect data, the data source where the data will be stored in the generated product, the type of geometry of the sensor, and a set of properties owned by the sensor, specified with the `Property` class. A property is defined by its name and its data type, which can be any of the types in the enumeration `SupportedType`.

The `Measurement` class defines the measurements that the sensor will collect over time, and it is a specialization of the `Observation` class from the OGC SensorThings API standard. A measurement is defined by its name and its data type which can be the same as the sensor property type. It also optionally defines its units as text and its icon, which can be any of the Material Design Icons[2]. A `Range` can be added to each measurement property so that the data in the final product can be shown in different colors depending on the ranges specified. Lastly, for each sensor, a bounding box can be defined (i.e., the `BBox`), so that the final map will start at a fixed position covering all sensors.

In order to build a datawarehouse with sensor data, we consider each `Sensor` with its `Measurements` to be a fact table of the datawarehouse. For each `Sensor`, multiple spatial and categorical dimensions can be defined. Each `SpatialDimension` has a name, a geometry, and a set of properties. It can also have a parent that contains a reference to another spatial dimension to create a hierarchical spatial dimension that can be used for aggregations and filters. Similarly, each `CategoryDimension` has a name and a corresponding field that represents the attribute that the final data warehouse will use to represent the categorical values. Finally, `SpatialGroup` and `CategoryGroup` associate each sensor with its spatial and categorical dimensions. The temporal dimension of the data is taken into account in the IoT dashboard, but it does not have to be specified by the user because a temporal hierarchical dimension is inferred by the framework from the interval attribute of the sensor.

Sensors can be grouped using the `SensorGroup` class. This allows the user to combine multiple sensors that measure certain attributes

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

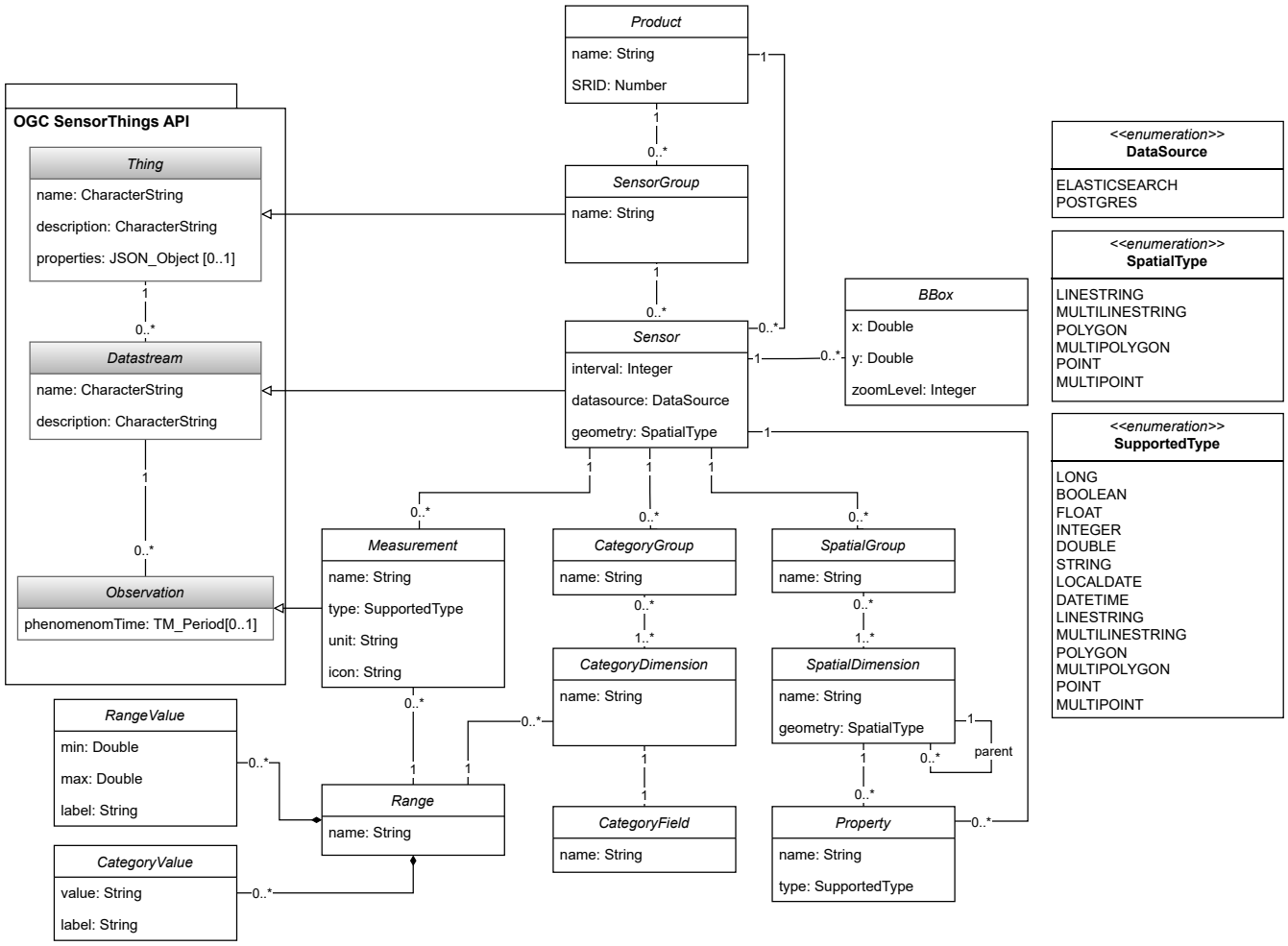

Figure 3: Class Diagram of the DSL Metamodel

of a physical or virtual entity, and hence it is a specialization of a `Thing` from the OGC SensorThings API standard. Finally, the class `Product` represents the complete project with a name and the spatial reference system (SRID).

The listings that follow show the textual syntax of our DSL. The specification of a product starts with the sentence `CREATE PRODUCT` (see Listing 1). Then, each sensor definition can be specified using the sentence `CREATE SENSOR` (see Listing 2). For each sensor type, the language allows to specify the sensor data, its properties (in the `WITH PROPERTIES` clause), its measurements (in the `MEASUREMENT DATA` clause), the spatial dimensions (in the `SPATIAL GROUP` clause), the categorical dimensions (in the `CATEGORICAL GROUP` clause), and the sensor network bounding box (in the `BBOX` clause). It is also possible to add a `RANGE` to `CATEGORICAL GROUP` or measurements so that the final user can aggregate and filter by these ranges.

```
CREATE PRODUCT name USING srid;
```

Listing 1: CREATE PRODUCT sentence

```
CREATE SENSOR name (
    interval: integer,
    datasource: datasourceType,
    geometry: geometryType
) WITH PROPERTIES (
    propertyName1 dataType1 [DISPLAY_STRING] [REQUIRED] [UNIQUE],
    propertyName2 dataType2 [DISPLAY_STRING] [REQUIRED] [UNIQUE]
    ...
) WITH MEASUREMENT DATA
    measurementName1 dataType1 [UNITS String] [ICON String] [RANGE
      rangeRef1],
    measurementName2 dataType2 [UNITS String] [ICON String] [RANGE
      rangeRef2],
    ...
) WITH SPATIAL GROUP dimAggName1 (
    spatialDimRef1,
    spatialDimRef2,
    ...
) WITH SPATIAL GROUP dimAggName2 (
    spatialDimRef3,
    ...
) WITH CATEGORICAL GROUP catAggName1 (
    categoricalDimRef1 [RANGE rangeRef1],
    ...
) WITH BBOX
```

```
    ([Double, Double], Integer) | (Double, Double, Integer)
);
```

Listing 2: CREATE SENSOR sentence

Spatial and categorical dimensions are defined using specific sentences so that they can be reused. Listing 3 shows the syntax of the `CREATE SPATIAL DIMENSION` sentence. Similarly, Listing 4 shows the syntax of the `CREATE CATEGORICAL DIMENSION` statement.

```
CREATE SPATIAL DIMENSION dimName (
    geometry: geometryType
) WITH PROPERTIES (
    prop1 dataType1 [DISPLAY_STRING]
    ...
) WITH PARENT (
    dimNameParent
);
```

Listing 3: CREATE SPATIAL DIMENSION sentence

```
CREATE CATEGORICAL DIMENSION dimName (
    field: String
);
```

Listing 4: CREATE CATEGORICAL DIMENSION sentence

The statement `CREATE SENSORGROUP` (shown in Listing 5) is used to define a collection of sensors.

```
CREATE SENSORGROUP sensorGroupId (
    SensorId1,
    SensorId2
);
```

Listing 5: CREATE SENSORGROUP sentence

Finally, `CREATE RANGE` can be used to define ranges, each one consisting of a name and a list of ranges. These ranges can either be made up of a single unique value or a range that is represented in the format `x TO y`. Additionally, each range can have a specific color assigned to it by using the `COLOR` keyword. Lastly, each range will have a label assigned to it using the `AS` keyword.

```
CREATE RANGE rangeName (
    propValue1 AS "propName1" [COLOR hexColor] ||
    FROM propValue1 TO propValue2 AS "propName1" [COLOR hexColor]
    ...
);
```

Listing 6: CREATE RANGE sentence

3.1.2 ***Feature Selection.*** Figure 4 depicts the feature model of the software product line. It enables users to select functionality related to the management of layers on the map, the types of legends used for data aggregation, available filters, the presence of a timeline, whether rasters and value evolution charts can be displayed, creation of producers and consumers for real-time data ingestion, and inclusion of an artificial data generator for development purposes.

3.1.3 ***Deployment Information.*** In addition to specifying the sensor information and product functionality, users can also choose to either generate only the source code of the final product or deploy the application on a machine. This can be achieved by specifying the desired deployment method, including running the complete system locally, remotely via SSH, or on Amazon Web Services (AWS).

Figure 4: Feature Model of sensor-related features

Based on this selection, users can input machine credentials for SSH connectivity, AWS credentials, and other AWS-related information such as the type of instance to create.

See the Product Deployment section 3.4 for additional details on the deployment process.

3.2 SensorPublisher

SensorPublisher is the tool that orchestrates the entire workflow. It checks the validity of the product description, generates a complete product specification, and uses the Sensor Builder SPL to generate the product source code. Finally, if necessary, it uses the Product Deployer to deploy the product. The tool can be used through both command line and graphical user interface (GUI) modes. In command line mode, the product description, comprising an instance of the sensor DSL, a feature selection, and deployment details are provided as text files. Alternatively, the tool offers a web-based GUI. Within the GUI, users can define sensor specifications using an embedded text editor, as shown in Figure 5, which incorporates a drag-and-drop functionality for text files. Additionally, a visual representation of the feature model is available. Users can customize the final product by selecting desired features from this representation, as shown in Figure 6.

3.3 Sensor Builder SPL

Once the user has completed the configuration process by choosing features from the feature model, using the DSL to describe the sensor network, and optionally choosing deployment information, SensorPublisher generates a specification. This specification is created in a JSON file containing all the data required to generate the product's final code.

We developed and annotated code for the SPL to support the generation of the products. This code base implements a large set

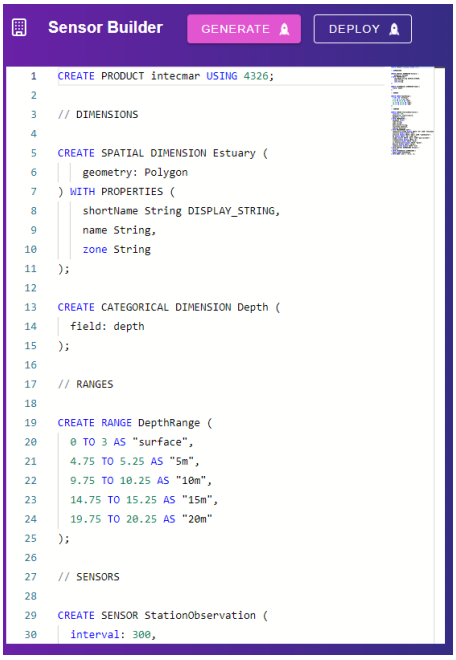

Figure 5: SensorPublisher GUI DSL Editor

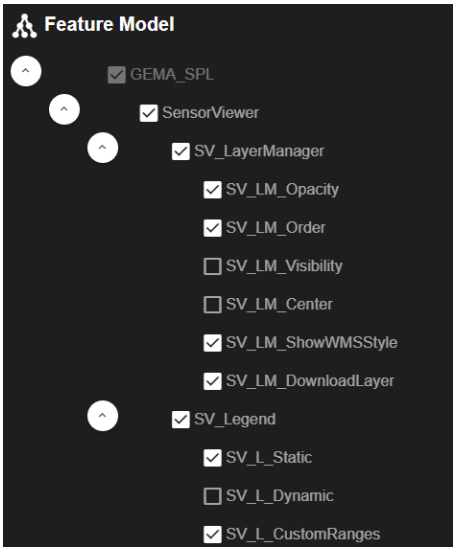

Figure 6: SensorPublisher GUI Feature Model Selector

of functionalities related to sensor information, including spatial, temporal, and categorical group data. Moreover, it enables visualization of information on a map, allowing the end users to perform analyses within a web-based product. The web-based product also allows for query creation, enabling aggregations and data filtering, as well as workflows for data ingestion into the final system. All the code is annotated to identify the features shown in Figure 4.

The general architecture of the annotated code (see Figure 7) consists of a front-end web-based application created with `Vue.js`, and

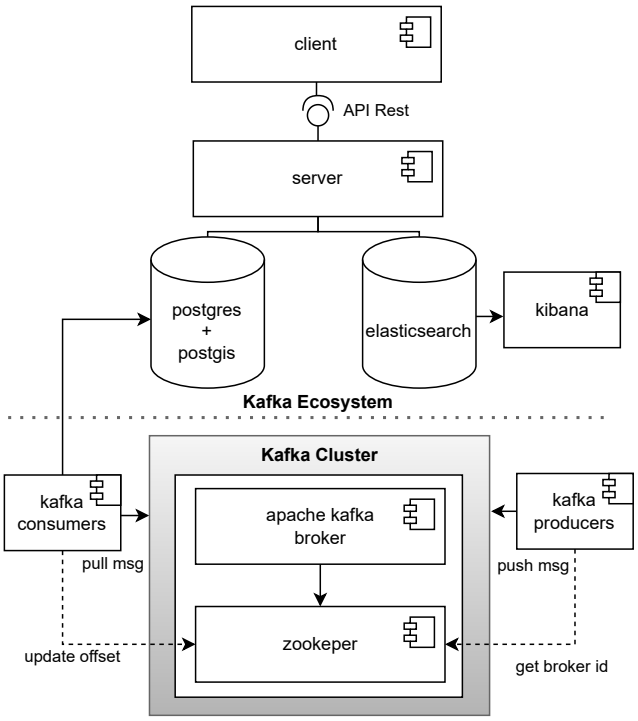

Figure 7: Generated Product Architecture

for map visualization, we developed a library named `map-viewer`, which serves as a wrapper for the Leaflet.js library [3]. The Vue web application renders the user interface and manages user interactions. It communicates with server-side components through RESTful APIs. The back-end infrastructure comprises a Spring Boot-based server responsible for data processing, business logic, and serving requests from the Vue client. It can retrieve, store, and manipulate data and it uses two different DBMS to store and retrieve sensor data: a relational DBMS (PostgreSQL) and a search engine (ElasticSearch). Data ingestion is achieved using Apache Kafka, a distributed data streaming platform enabling users to publish, subscribe to, store, and process streams of records in real-time. It provides a distributed messaging system where producers publish records into topics, and consumers subscribe to these topics to consume the records. In our particular case, the producers and consumers are defined as follows:

- **Kafka Producers:** Spring Boot-based applications responsible for collecting and publishing GIS-related data to Kafka topics. They gather data from external data providers or IoT devices and publish it to Kafka topics for further processing.
- **Kafka Consumer:** An individual Spring Boot application subscribed to all Kafka topics to handle data persistence, indexing, and updating to ensure data availability and consistency. It processes incoming messages and stores the data in the relevant databases.

The architecture of the system provides scalability, maintainability, extensibility, and seamless interaction between the databases

[3] https://leafletjs.com/

(PostgreSQL, ElasticSearch). The Kafka messaging system, in addition, guarantees a decoupled approach to data ingestion, allowing for fault tolerance and scalability by isolating the data insertion procedure from the client/server communication.

In IoT domains, systems often face the challenge of processing substantial amounts of sensor measurement data efficiently. This includes not only analyzing historical data promptly but also ingesting and visualizing data almost instantaneously. To address this aspect of the architecture, we opted for ElasticSearch, a distributed RESTful search and analytics engine. Its capabilities in processing, querying, and storing large volumes of data efficiently make it an ideal choice. The objective behind using ElasticSearch is to enhance the end-user experience by providing rapid responses to queries concerning large-scale sensor measurements.

Given that each sensor `Measurement` includes a temporal value (i.e., `phenomenonTime`), our product generates an index for each day of the year, using the sensor name and date as the index name (e.g., `trafficSensorYYYYMMdd`). This method guarantees that daily measurements are stored in different indexes, ensuring scalability and preventing performance degradation over time. ElasticSearch allows query execution across a range of indexes using regular expressions, enabling the system to construct aggregation and search queries easily.

For the product derivation, we use a library called **spl-js-engine**. 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 the generation [4]. At the end of this step Sensor Builder SPL creates a final product with all the functionality that the user needs.

3.4 Product Deployer

As part of the workflow, SensorPublisher provides the option to deploy the generated product. Docker[4], an open platform for creating, distributing, and executing applications, is used to deploy the entire system. Docker allows developers to deliver applications faster by separating them from infrastructure. All the components of the final products are decoupled into different docker components that connect between them using docker-compose. The deployment options provided `Product Deployer` include:

- **Local**. For local deployment, Product Deployer ensures the smooth installation of the product onto the user's device, providing a practical and easily accessible environment for testing.
- **SSH**. Users can also choose to deploy the product on Ubuntu/Debian machines via SSH. This option for distributed deployment offers greater accessibility and flexibility to a wider range of users. The Product Deployer takes care of preparing the machine for deploying the final product by installing Docker and configuring Docker Compose if not already installed on the machine.
- **AWS**. For deployment on Amazon Web Services (AWS), users may opt for enhanced accessibility and scalability. Product Deployer facilitates the deployment process by creating an AWS instance using user credentials, configuring and installing Docker on the instance, copying the source code of the product, and utilizing `docker-compose` to run it.

[4]https://www.docker.com/

4 USAGE SCENARIOS

This section explores three distinct scenarios where we have used SensorPublisher to create IoT dashboards. These scenarios focus on analyzing meteorological conditions, traffic and air quality data, and monitoring seawater properties.

4.1 Meteorological Monitoring System

Based in Galicia, Spain, MeteoGalicia is a weather prediction service. It offers forecasts and meteorological data for the area, including information on temperature, precipitation, wind speed, and atmospheric conditions. MeteoGalicia provides precise and current weather forecasts to help citizens, companies, and government agencies in making well-informed decisions. We got from MeteoGalicia a dataset with information from sensors that measure precipitation, pressure, sun radiation, humidity, temperature, and wind. In addition to showing data at the individual sensor level, we define a spatial dimension that aggregates sensors at the Council and Province levels in Galicia. This spatial dimension allows the assessment of meteorological data across different regions, allowing users to uncover local weather patterns. Listing 7 shows the DSL instance that defines the sensor data model.

```
CREATE PRODUCT meteorological USING 4326;

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

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

CREATE SENSOR MeteorologicalStation (
    interval: 500,
    datasource: elasticsearch,
    geometry: Point
) WITH PROPERTIES (
    name String,
    registrationDate DateTime
) WITH MEASUREMENT DATA (
    precipitation Double UNITS "L/m2",
    pressure Double UNITS "hPa",
    solarRadiation Double UNITS "W/m2",
    relativeHumidity Double UNITS "%",
    temperature90 Double UNITS "C",
    windDirection Double UNITS "",
    windSpeed Double UNITS "m/s"
) WITH SPATIAL GROUP Administrative (
    Province, Council
) WITH BBOX ([42.7, -8.1], 9);
```

Listing 7: Case Study DSL definition - Meteorological Status in Galicia Province

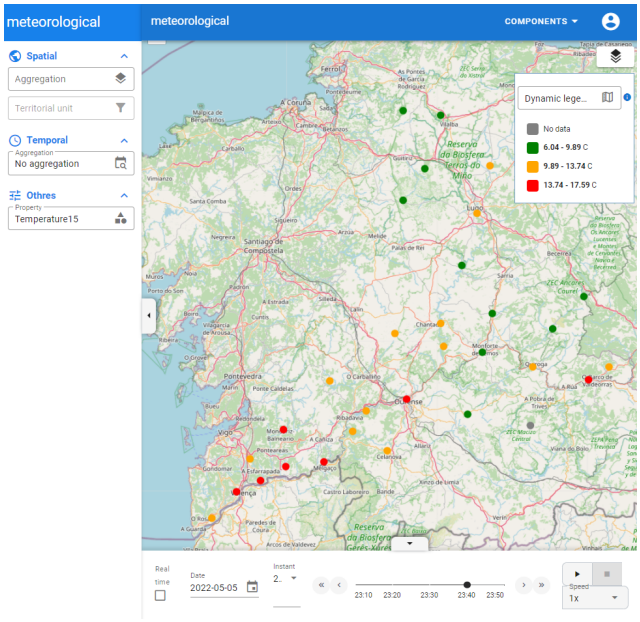

Figure 8: Meteorological Product - Map with Sensors

The finished meteorological product is shown in a screenshot in Figure 8. To make it easier to navigate between the various properties that the sensors measure, aggregations, and filters for the spatial dimension are displayed on the left side of the screen along with the properties themselves. The sensors are positioned on a map that is shown in the middle of the screen, and the values of each sensor's measurement property are displayed in color based on the map's legend. To navigate in the temporal dimension, a navigation time bar is shown in the bottom center of the screen.

4.2 Traffic and air quality data

The city of Madrid provides citizens with access to a range of open data concerning the city's status [5] [6]. In this scenario, our objective is to analyze the traffic and air quality within the city, with the ability to aggregate and filter the data based on census sections, districts, and municipalities.

In this scenario, we have two different sensor definitions: one for traffic sensors and another for air quality sensors. These two sensor types will share a spatial dimension that reflects the administrative division of the city (i.e., Section, District, and Municipality). Furthermore, an extra spatial dimension known as *Voronoi* will be added to the air quality sensors, which will help analyze pollution hotspots inside the city that are not connected to departmental boundaries. The traffic sensors collect measurements of traffic intensity, average vehicle speed, and lane occupancy percentage. Meanwhile, the air quality sensors collect measurements related to air pollution, including the levels of carbon dioxide, carbon monoxide, pollution microparticles, and so forth. Listing 8 shows the DSL instance that defines the sensor data model.

[5] https://datos.madrid.es/portal/site/egob
[6] https://informo.madrid.es/

```
CREATE PRODUCT  magist USING SRID 4326;

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

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

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

CREATE SPATIAL DIMENSION Voronoi (
    geometry: Geometry
) WITH PROPERTIES (
    magnitude Integer DISPLAY_STRING
);

CREATE SENSOR QAObservation (
    interval: 3600,
    datasource: elasticsearch,
    geometry: Point
) WITH PROPERTIES (
    address String,
    station Long,
    registrationDate DateTime,
    nameType String,
    viaName String
) WITH MEASUREMENT DATA (
    so2 Double,
    co Double,
    no Double,
    no2 Double,
    pm25 Double,
    pm10 Double,
    nox Double
    ...
) WITH SPATIAL GROUP Administrative (
    Section, District, Municipality
) WITH SPATIAL GROUP Voronoi (
    Voronoi
) WITH BBOX ([40.42, -3.7], 12);

CREATE SENSOR TrafficObservation (
    interval: 300,
    datasource: elasticsearch,
    geometry: Point
) WITH PROPERTIES (
    description String,
    subarea Long,
    name String
) WITH MEASUREMENT DATA (
    intensity Double UNITS "veh/h" ICON "network_check",
    velocity Double UNITS "km/h" ICON "speed",
    ocupation Double UNITS "%" ICON "emoji_transportation"
) WITH SPATIAL GROUP Administrative (
    Section, District, Municipality
```

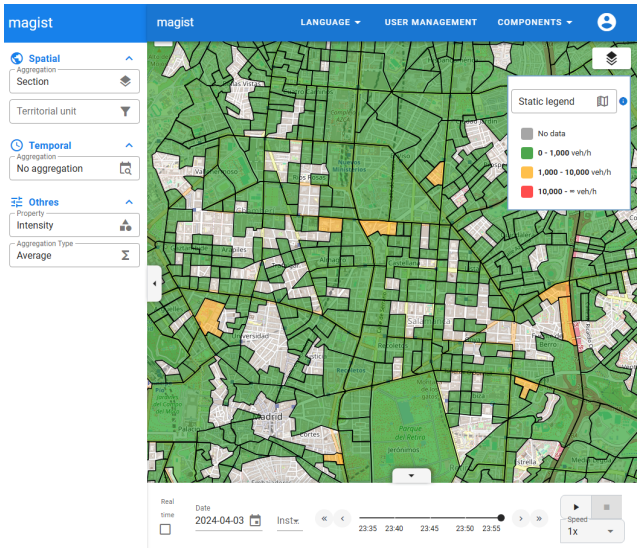

Figure 9: Traffic Product - Spatial Aggregation by District

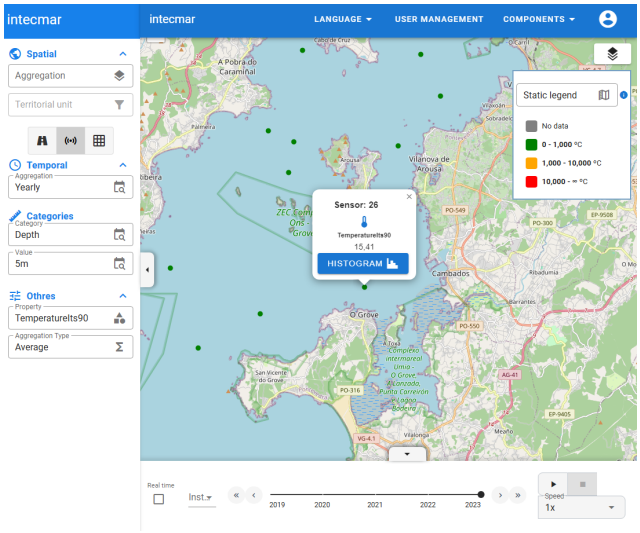

Figure 10: Marine Product - Sensor with Categorical Dimension

```
) WITH BBOX ([40.42, -3.7], 12);
```

Listing 8: Case Study DSL definition - Traffic and Quality Air in Madrid

A screenshot of the administrative spatial dimension (defined by the sentence `WITH SPATIAL GROUP Administrative` in Listing 8) can be seen in Figure 9. The tool enables the spatial aggregation and filtering of sensors grouped by district, enabling the analysis of observations in these areas.

4.3 Seawater monitoring

The Galician Technological Institute for the Control of Marine Environment (INTECMAR) provides marine environment data to geosciences researchers. One dataset includes measurements from CTD sensors. CTD stands for conductivity, temperature, and depth sensor, a common instrument used in oceanography to measure various properties of seawater with depth. INTECMAR conducts measurements along Galicia's coasts using sensors integrated in maritime buoys. These sensors detect parameters such as water temperature, salinity, pH, and oxygen content, alongside conductivity, UV fluorescence, and radiance, measured at specific water depths. Spatial filtering can be applied to these sensors using an "Estuary" spatial dimension. Additionally, each sensor measurement is associated with a specific water depth, requiring a categorical dimension for the "depth" field, categorized into specific ranges. Listing 9 shows the DSL instance that defines the sensor data model.

```
CREATE PRODUCT intecmar USING 4326;

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

CREATE CATEGORICAL DIMENSION Depth (
    field: depth
);

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"
);

CREATE SENSOR StationObservation (
    interval: 300,
    datasource: elasticsearch,
    geometry: Point
) WITH PROPERTIES (
    maxDepth Float,
    code String,
    name String,
    description String,
    startTime DateTime
) WITH MEASUREMENT DATA (
    temperatureITS90 Double UNITS "C" ICON "thermometer-low",
    salinity Double UNITS "PSU",
    pressure Double UNITS "dbar" ICON "speedometer",
    ph Double UNITS "pH" ICON "ph",
    oxygen Double UNITS "mg/l" ICON "gas-cylinder",
    transmittance Double UNITS "m",
    irradiance Double UNITS "W/m2",
    uv_flourescence Double UNITS "mg/m3",
    density Double UNITS "kg/m3",
) WITH SPATIAL GROUP Estuary (
    Estuary
) WITH CATEGORICAL GROUP Depth (
    Depth RANGE DepthRange
) WITH BBOX ([42.7, -8.1], 9);
```

Listing 9: Case Study DSL definition - Marine Area Monitoring System

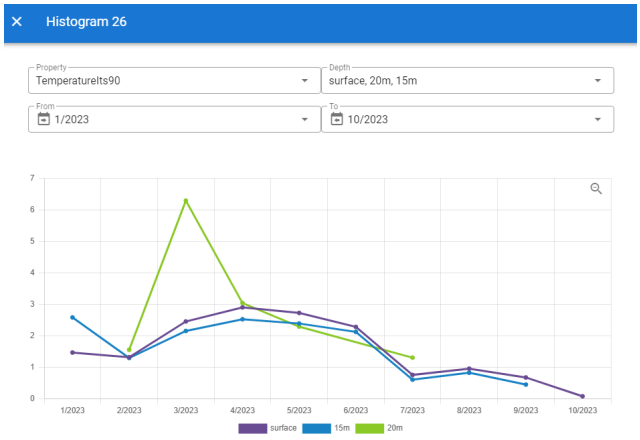

Figure 11: Marine Product - Value evolution chart by Categorical Dimension "depth"

The categorical dimension "depth", which is defined by the user in Listing 9 with the sentence `CREATE CATEGORICAL DIMENSION depth`, appears on the left bar in Figure 10. Now that this new dimension has been added, the user may perform analyses and filter the results based on depth ranges. Additionally, if a spatial or temporal aggregation is chosen, the popup can now display an evolution chart that shows the values of each range of the categorical dimension. This is displayed in Figure 11, which aggregates measurements monthly and distinguishes between depth dimension range values.

5 CONCLUSIONS AND FUTURE WORK

We have presented a framework that offers a complete way to deal with the difficulties of developing sensor-based, web-centric, data warehousing information systems in geosciences domains. The software product line, the domain-specific language, and the low-code development tool simplify the development process and increase its efficiency and accessibility for geoscientists. Our approach empowers geoscience researchers by enabling them to share their datasets through a web application with little effort.

The first outcome of our research is the successful definition and implementation of a software product line tailored specifically for IoT dashboards within the geoscience domain. The products generated allow users to filter and aggregate data spatially, temporally, and by attribute values. Users can also create charts (e.g., histograms and value evolution charts). Finally, the products are efficient in terms of querying and retrieving large datasets and an ingestion subsystem is provided to receive and store sensor data streams. We showcased and validated the software product line in three real-world geoscience scenarios: a meteorological monitoring system, traffic and air quality data, and seawater monitoring. Initial feedback from geoscience researchers who assisted in setting up the scenarios indicates that our software product line significantly reduces the time and effort required to establish IoT-based systems for their research. Rather than starting from scratch, researchers can now focus their efforts on aspects specific to their applications, leading to increased productivity. As part of our future work, We aim to conduct formal usability evaluations to gather comprehensive feedback and insights from geoscience researchers.

The second outcome is the definition and implementation of a DSL designed for describing data warehouses of sensor data, which is also aligned with the Open Geospatial Consortium (OGC) SensorThings API standard. Using this language, researchers and developers can define the various sensor types within their network, including detailed descriptions of properties and measurements associated with each sensor type. Additionally, the language enables researchers to specify the dimensions of their data warehouse (both spatial and attribute-based) and to describe visualization styles for their data (i.e., assigning ranges of values and colors to each sensor measurement). The integration of the language into the software product line increases flexibility and reduces the effort required for defining and configuring data warehouses of sensor data.

The third outcome is the development of a user-friendly, low-code tool aimed at geoscience researchers that simplifies the process of publishing collected datasets. The tool generates and deploys web-based applications so that geoscientists with little programming experience do not have to spend a lot of time manually coding.

Finally, we believe that our framework can be replicated in other scenarios. While the SPL and the DSL would have to be defined for those scenarios, the framework has been defined and implemented in a sufficiently general manner to enable easy reuse.

In future work, we want to integrate new types of IoT sensors into our framework. Our current implementation focuses on static sensors, but we want to include mobile sensors by extending the DSL to be able to define these types of sensors, and the software product line to include functionality to query and visualize them. As another part of our future work, we're actively working to release our framework as open-source software. We have already published some of the basic components (i.e., the derivation engine spl-js-engine[7], the map-viewing component[8], a DSL for web-based geographic information systems that is used as a foundation for our DSL[9], and the code uploading component[10]).

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]; [FLATCITY-POC: PDC2021-121239-C31]; [SIGTRANS: PDC2021-120917-C21]; partially funded by MCIN/AEI/10.13039/501100011033 and EU/ERDF A way of making Europe: [EarthDL: PID2022-141027NB-C21]; partially funded by GAIN/Xunta de Galicia: [GRC: ED431C 2021/53]

REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2013. FAMILIAR: A domain-specific language for large scale management of feature models. *Science of Computer Programming* 78, 6 (2013), 657–681. https://doi.org/10.1016/j.scico.2012.12.004 Special section: The Programming Languages

[7] https://www.npmjs.com/package/spl-js-engine
[8] https://www.npmjs.com/package/@lbdudc/map-viewer
[9] https://www.npmjs.com/package/@lbdudc/gp-gis-dsl
[10] https://www.npmjs.com/package/@lbdudc/gp-code-uploader

track at the 26th ACM Symposium on Applied Computing (SAC 2011) Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments.

[2] Suilen H. Alvarado, Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, and Ángeles S. Places. 2020. Developing Web-based Geographic Information Systems with a DSL: Proposal and Case Study. *Journal of Web Engineering* 19, 2 (Jun. 2020), 167–194. https://doi.org/10.13052/jwe1540-9589.1923

[3] Don Batory, Clay Johnson, Bob MacDonald, and Dale von Heeder. 2002. Achieving extensibility through product-lines and domain-specific languages: a case study. *ACM Trans. Softw. Eng. Methodol.* 11, 2 (apr 2002), 191–214. https://doi.org/10.1145/505145.505147

[4] Alejandro Cortiñas, Miguel R. Luaces, and Óscar Pedreira. 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* (Graz, Austria) *(SPLC '22)*. Association for Computing Machinery, New York, NY, USA, 66–69. https://doi.org/10.1145/3503229.3547035

[5] Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, Ángeles S. Places, and Jennifer Pérez. 2017. Web-based Geographic Information Systems SPLE: Domain Analysis and Experience Report. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume A* (Sevilla, Spain) *(SPLC '17)*. Association for Computing Machinery, New York, NY, USA, 190–194. https://doi.org/10.1145/3106195.3106222

[6] David de Castro, Alejandro Cortiñas, Victor Lamas, and Miguel R. Luaces. 2023. GIS-Publisher: From a Geographic Data Set to a Deployed Product with One Command. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume B* (Tokyo, Japan) *(SPLC '23)*. Association for Computing Machinery, New York, NY, USA, 20–24. https://doi.org/10.1145/3579028.3609009

[7] Marjan Mernik, Jan Heering, and Anthony M. Sloane. 2005. When and how to develop domain-specific languages. *ACM Comput. Surv.* 37, 4 (dec 2005), 316–344. https://doi.org/10.1145/1118890.1118892

[8] Open Geospatial Consortium. 2021. OGC SensorThings API Part 1: Sensing Version 1.1. Technical Specification. https://www.ogc.org/standards/sensorthings

[9] Markus Voelter and Eelco Visser. 2011. Product Line Engineering Using Domain-Specific Languages. In *2011 15th International Software Product Line Conference*. 70–79. https://doi.org/10.1109/SPLC.2011.25