A Domain Specific Language for Web-based GIS

Suilen H. Alvarado, Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira and Angeles S. Places
Universidade da Coruña, Centro de Investigación CITIC, Laboratorio de Bases de Datos, Facultade de Informática, Campus de Elviña s/n, 15071, A Coruña, Spain

Keywords: Model-driven Engineering, Domain Specific Language, Geographic Information Systems.

Abstract: Geographic information systems (GIS) manage entities with a spatial component (typically, in the form of a point, line, or polygon defined according to a known geographic coordinate system), and provide specific operations to process them, and specific interfaces to visualize them. Although different GIS may provide a completely different set of functionalities, they all share a common set of concepts, architecture, and technologies. Therefore, in the development of GIS there are typically large parts of the system that can be very repetitive. In this paper, we present a domain-specific language to develop GIS from high-level declarative specifications. We present a metamodel for web-based GIS, and a domain-specific language based on that metamodel. We also present a usage example that shows how the language would be used in a real scenario.

1 INTRODUCTION

Geographic Information Systems (GIS) allow to collect, integrate, query, analyze, and represent data regarding geospatial information (Worboys and Duckham, 2004; Longley et al., 2015). The data managed in a GIS typically represent real-world entities with a geospatial component, such as points of interest, roads, trajectories, or electrical networks, for example. In general, GIS are developed focusing on solving planning and management problems that cover several areas of application. Some fields in which using a GIS solution seems mandatory nowadays are, for example, transportation management, environmental management, territorial planning, logistics, infrastructures, or security. Moreover, the rise of mobile technologies and devices with affordable geolocalization capabilities has broadened the range of applications of GIS, and also the volumes of data they manage.

Independently of the specific purpose and application domain of each GIS, these systems share a set of common components and features such as digitizing geographic data, showing geospatial data in map viewers, common tools related to maps, route optimization, etc. Currently, GIS are developed using technologies, tools, and libraries that facilitate the storage and processing of geospatial information. However, they are typically developed by implementing the system "from scratch", or reusing basic components. In the last years, the standards from the Open Geospatial Consortium (OGC)[1], and the general rise of web applications have driven the evolution from desktop-based to web-based GIS, which are nowadays the most common solution. Given the similarity, and the number of elements shared between different GIS, we believe that model-driven development techniques can be applied in this domain.

Model-driven engineering (MDE) aims at using system models as active artifacts in the development, successively transforming them into models at a lower level of abstraction and, finally, into the source code of the system (Pastor and Molina, 2007; Brambilla et al., 2017). In this way, the effort required to develop the system (or a part of it) is smaller, and the code generated automatically from high-level models of the system is less likely to present errors. The first step in MDE is to develop a metamodel of the domain, which defines a *domain-specific modeling notation*. The metamodel can also be used to define a *domain-specific language* (DSL), that allows specifying the system in a language at a higher-level of abstraction than that of general-purpose programming languages, with a significantly smaller development effort.

In this paper, we explore the application of MDE techniques to the development of web-based GIS, and we propose a DSL for GIS development. As we will see in Section 3, it is a high-level declarative language that allows the developer to define, at a high level of abstraction, the elements that will be managed in the

[1] OGC: http://www.opengeospatial.org/

Alvarado, S., Cortiñas, A., Luaces, M., Pedreira, O. and Places, A.
A Domain Specific Language for Web-based GIS.
DOI: 10.5220/0008559104620469
In *Proceedings of the 15th International Conference on Web Information Systems and Technologies (WEBIST 2019)*, pages 462-469
ISBN: 978-989-758-386-5

GIS, and how they are going to be visualized.

The rest of the article is structured as follows: In Section 2 we present background and related work. In Section 3 we describe the main concepts, elements, and characteristics of GIS, and we present a metamodel for web-based GIS and a DSL that allows automating the development of this type of applications. In Section 4 we present a case of use of the DSL where we specify a real application. Finally, Section 5 presents the conclusions of the paper and lines for future work.

2 BACKGROUND AND RELATED WORK

The rapid evolution of new technologies, the increasing complexity of software, and the demand to satisfy a greater number of requirements are some of the motivations of paradigms that aim at automating the software development process. *Model-driven engineering* (MDE) is a software development approach that promotes the use of models as active artifacts in all stages of software development. In MDE, high-level models are automatically transformed into models at a lower level of abstraction, and finally, into the source code (or part of) the system. The goal of this paradigm is to produce software following an approach similar to that of other traditional industries. One of the approaches within MDE is *model-driven development* (MDD) (Brambilla et al., 2017), which defines models as the main artifact for modeling software systems at a level of abstraction higher than the allowed by programming languages. The objective of this approach is to increase the level of automation, improve aspects of quality, productivity and reduce the costs associated with the processes of evolution and maintenance (Pons et al., 2010).

A standard defined by the Object Management Group (OMG)[2] on its particular vision of MDD is the *model-driven architecture* (MDA)[3] (Pastor and Molina, 2007), structured in four layers: CIM (computational independent models), PIM (platform-independent models), PSM (platform-specific models), and ISM (implementation-specific model). These models can be defined using general-purpose modeling languages, such as UML, or *domain-specific languages* (DSL), designed to be used in particular application domains.

A DSL is a language designed for software development in a specific application domain. The difference between a DSL and a general-purpose programming language is that a DSL allows us to work directly with domain-specific concepts and constructs, which leads to a greater expressiveness (Mernik et al., 2005; Fowler, 2010). Although implementing a DSL can require a significant effort, their main benefit is that they allow us to specify/implement a system with significantly less effort.

A wide variety of DSLs have been developed in different application domains. For example, in *software engineering*, there has been proposed a DSL to support the process of generating source code for desktop-based database application using the Java language (Lolong and Kistijantoro, 2011), another DSL to model performance tests for web applications (Bernardino et al., 2016), or even to specify textual use cases and, from them, generate semi-automatically use case, class and sequence diagrams (Miranda et al., 2017). Within the area of *internet of things* (IoT), a *visual domain-specific modeling language* (VDSML) was defined to abstract IoT application designers from some complexities that present this type of application, like the wide range of hardware and software entities, or the middle-ware specific features (Salihbegovic et al., 2015). Also in this field, another DSL was proposed to model a smart city system based on specific domain concepts (Rosique et al., 2017). Finally, in *mobile application development*, there are DSLs to implement systems for multiple platforms (Kramer et al., 2010; Ribeiro and da Silva, 2014), or to describe the structure and behavior of real-time mobile applications centered on data (Behrens, 2010).

The systematic mapping presented in (Kosar et al., 2016) highlights some lines of DSL research that may be further studied. For example, this mapping revealed that most of the articles are focused on the design and implementation of DSLs, but few of them considered aspects such as validation and usability evaluation, domain analysis, or maintenance.

In previous works (Cortiñas et al., 2017b; Cortiñas et al., 2017a), we explored the automated development of GIS through a combination of *software product line* (SPL) technologies and basic MDE techniques applied to the generation of the database and data model. Our platform allowed the user to define the data model of the system, and to specify a selection of optional features that could be included in the final system. In this paper we further explore the application of MDE techniques for the development of web-based GIS through the definition of a DSL that considers the definition of the domain geospatial entities, and also how they will be visualized in the web.

[2]OMG: http://www.omg.org

[3]MDA: http://www.omg.org/mda

3 A DOMAIN SPECIFIC LANGUAGE FOR WEB GIS

The first GIS applications were developed for desktop environments. At that time, the capabilities of web technologies were very limited, and the requirements of a GIS were too complex for web environments, so desktop was the only choice. However, the improvements in web technologies and the computational power of current devices (server, desktop, and mobile) have made it possible to develop web-based GIS. Although some GIS applications are still developed for desktop environments, the web has become the preferred choice. The ISO and the Open Geospatial Consortium (OGC) have defined a set of evolving standards which define most of the aspects for GIS domain, including models, procedures, services, and architectures. We can also see the focus on the web in these standards since many network-based services were defined, such as the *web map service*[4] (WMS) or the *web feature service*[5] (WFS).

In this section, we present a DSL for web-based GIS. The main characteristics of the DSL are *(i)* it allows the developer to specify the entities to be managed in the GIS, and how they will be visualized in the web through layers and maps, *(ii)* it is a declarative language, that is, using this DSL the developer just specifies the data model of the system and how the data will be shown, without having to implement any details related to these features. In sub-section 3.1 we briefly present the architecture of a GIS and a metamodel of this domain. In sub-section 3.2 we present the DSL.

3.1 GIS Architecture and Main Constructs

The main difference between a regular information system and a GIS is the support of geospatial data types. That is, the data model of a GIS can include properties such as *Point*, *Line*, or *Polygon*. A *Point* is defined by a *latitude* and *longitude*, and represents a particular position in the space. It is used to locate any objects in the space, such as traffic lights. A *Line* is a set of joined *Points*, and it is common using it to represent roads or pipes. *Polygons* are used to represent areas, such as administrative divisions of the territory. There is also a data type that can represent any kind of geometry, called *Geometry*, which is also in fact a *superclass* of all the rest.

Due to the nature of these data types, there are specific operations we can execute on them. For example, we can check if two geometries intersect by using the spatial predicate `st_intersects`, or we can calculate the area of a *Polygon* by using the operation `st_area`. Similarly, we can create a new polygon by joining two existing polygons.

An important characteristic of the spatial data types is that they need to be defined within a *spatial reference system* (SRS). A SRS defines the map projection used by some spatial data or by a map viewer. Setting a specific SRS is required since there is not only one way to make the transformation between some coordinates and the actual position of an object. Depending on the spatial context for which a GIS is built, we may prefer to use one SRS or another. For example, if our product needs to handle data from all around the world, we need to use a global SRS such as WGS84, identified by the SRID 4326; but if we need to implement a product handling data from a smaller territory and in a very precise way, we would probably use a local SRS, such as the ETRS89 or UTM zone 29N, identified by the SRID 25829. The SRS are defined in the standard ISO 19111:2007[6].

Supporting geographic data types and spatial operations is done by using a specific set of tools and technologies that comply with the GIS standards. In this way, we have extensions for relational databases that handle GIS related features such as PostGIS[7], an extension for PostgreSQL, or Oracle Spatial[8]. If we want to handle these features in a higher-level language, like Java, we have tools such as the *Java Topology Suite* (JST) or the whole library collection *Java GeoTools*. The same operations can be run on JavaScript by using GeoJSON and libraries such as Turf. Finally, the view layer is built with the help of tools such as *OpenLayers* or *Leaflet*.

The visualization of geospatial data involves three concepts: layers, styles, and maps. A *layer* is an image that can be geographically bounded. This image can be composed by a set of real photos, such as in the case of a satellite view of the world, or it can be generated from some geographic data by applying a given *style*. When a *layer* is loaded in a map viewer, the viewer is responsible for asking the concrete image needed depending on the actual bounds of the view, using a specification such as TileLayer or WMS. There are cases where the image is generated by the map viewer itself when we are dealing

[4] *OpenGIS Web Map Server Implementation Specification*: http://www.opengeospatial.org/standards/wms

[5] *OpenGIS Web Feature Service 2.0 Interface Standard*: http://www.opengeospatial.org/standards/wfs

[6] ISO: *Geographic information: Spatial referencing by coordinates*: https://www.iso.org/standard/41126.html

[7] PostGIS: https://postgis.net/

[8] Oracle Spatial: https://www.oracle.com/database/technologies/spatialandgraph.html

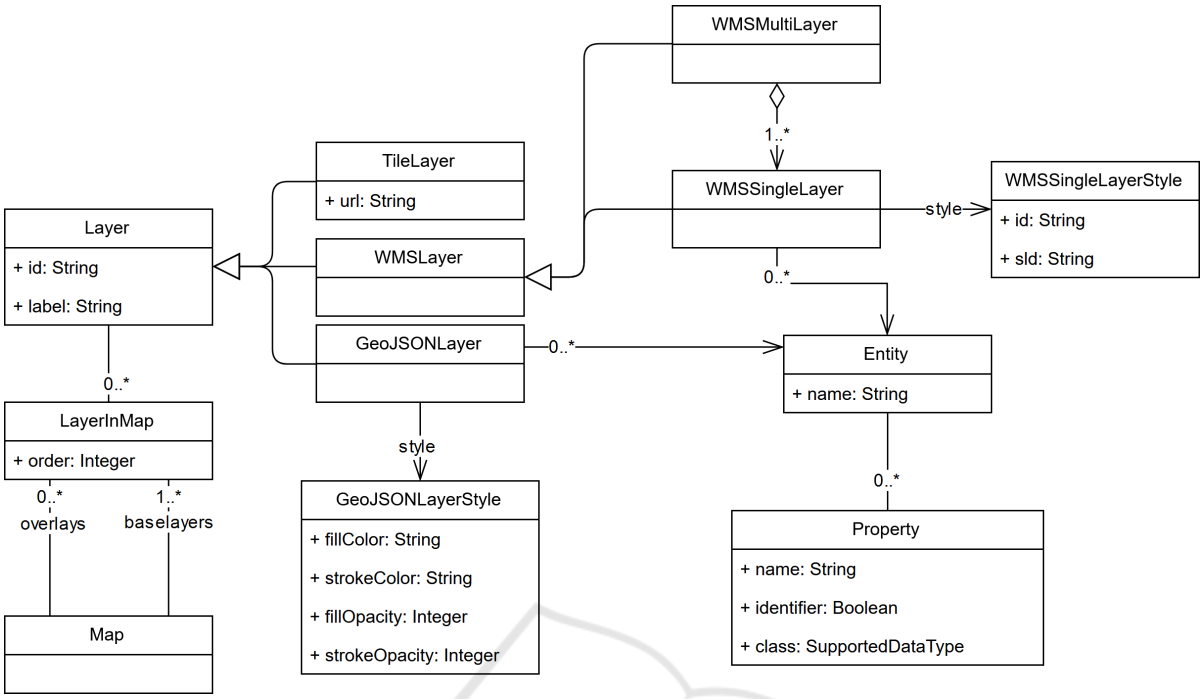

Figure 1: A metamodel of web-based GIS (reduced version).

with raw data loaded with GeoJSON documents. The *styles* determine how the data behind a *layer* is transformed into real images. Depending on the type of *layer*, we have different *style* specifications. For example, styles are not necessary for satellite images. If we are handling a WMS *layer* we need to use a *style layer descriptor* (SLD). A *map* is composed of a set of *layers* with their *styles*, rendered in a particular order. Usually the *layers* are generated from data of the application itself, this is, from *entities* which have a geographic property that represents their position. For example, we can have a *layer* of the traffic lights of a city, or a *layer* that shows the roads of a region. The metamodel presented in Figure 1 formalizes these concepts and how they relate to each other.

Figure 2 shows our architecture for a web-based GIS. The server side provides two services for the clients: a REST service handles most of the alphanumeric data and can provide geographic data in a serializable format (such as GeoJSON), and a WMS that provides cartography images by using a map server. Both the data and cartography services are fed from the same database. In the client side, the data layer is the component in charge of handling the REST communication, the logic of the application is handled by JavaScript code, and the templates are created using HTML. There is also a map viewer library that is working as a closed component and that can handle direct communication with the WMS.

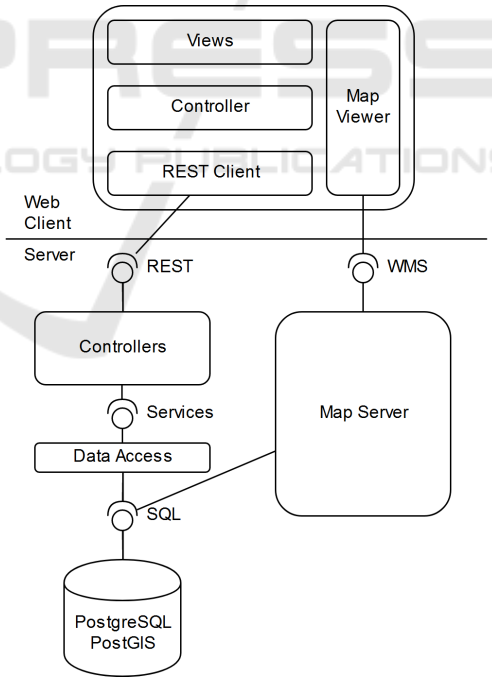

Figure 2: Typical architecture of a web-based GIS.

3.2 A DSL for GIS

In this section, we present and describe an excerpt of the declarative DSL we have designed for defining and generating web-based GIS.

The specification of a new application starts with the sentence `CREATE GIS`. In this sentence, we specify the name of the project, and the spatial reference system we will use (each reference system is defined by a specific id, `srid`). The DSL is thought to be used in an interactive terminal. The `USE GIS` sentence allows us to change from one project to another (see Listing 1).

```
CREATE GIS name USING srid;

USE GIS name;
```

Listing 1: `CREATE` and `USE` sentences.

The data model can be specified using the `CREATE ENTITY` sentence (see Listing 2). Each entity has a name and a set of properties. A property is defined also by a name and a data type. The type can be a boolean (`BOOLEAN`), a numeric type (`INTEGER`, `LONG`, `FLOAT` or `DOUBLE`), a alphanumeric type (`STRING` or `TEXT`), a temporal type (`DATETIME` or `DATE`), or any of the geographic types (`GEOMETRY`, `POINT`, `MULTIPOINT`, `LINE`, `MULTILINE`, `POLYGON` or `MULTIPOLYGON`). Each entity must have an identifier, defined by adding the keyword `IDENTIFIER` to the properties it is composed of. Relationships between entities can be defined as well, indicating the name of the relationship, its cardinally and whether it is bidirectional or just navigable in one of the directions.

```
CREATE ENTITY entityName (
  propertyName1 dataType1 [ IDENTIFIER ] [ REQUIRED ],
  propertyName2 dataType2 [ IDENTIFIER ] [ REQUIRED ],
  ...
  relationshipName1 entityName RELATIONSHIP{ (
    { 0..1 | 1..1 | 0..* | 1..* },
    { 0..1 | 1..1 | 0..* | 1..* }
  ) [ BIDIRECTIONAL ] | MAPPED_BY
     relationshipNameInTheOtherEntity },
  ...
);
```

Listing 2: `CREATE ENTITY` sentence.

Once the data model of the system has been defined, we can define the different layers available to be visualized in the map viewers of the application (see Listing 3). The `CREATE LAYER` sentence allows us to create three different types of layers: Tile Layers, WMS Layers and GeoJSON Layers. Tile layers are defined by an external URL, and they are used normally as base layers. GeoJSON layers are generated from an entity of the application, which is loaded into the map from the REST service applying a certain style. Entities loaded using this kind of layers can be editable using the forms of the application. Finally, WMS layers are loaded as cartography through a map server, and they can be generated from one or several entities from the application.

```
CREATE TILE LAYER name [ AS label ] (
  url STRING
);

CREATE GEOJSON LAYER name [ AS label ] (
  entity [ EDITABLE ],
  fillColor HEX,
  strokeColor HEX,
  fillOpacity FLOAT,
  strokeOpacity FLOAT
);

CREATE WMS_STYLE name (
  styleLayerDescriptor FILE_NAME
);

CREATE WMS LAYER name [ AS label ] (
  entity1 WMS_STYLE,
  entity2 WMS_STYLE,
  ...
);
```

Listing 3: `CREATE LAYER` sentence.

Finally, the sentence `CREATE MAP` (see Listing 4) allows us to define the map viewers of the application. Each map viewer has a set of layers, usually at least of them works as base layers, and then there is a set of overlays. Some of the layers can be hidden by default.

```
CREATE [ SORTABLE ] MAP name [ AS label ] (
  layer1 [ IS_BASE_LAYER ] [ HIDDEN ],
  layer2 [ IS_BASE_LAYER ] [ HIDDEN ],
  ...
);
```

Listing 4: `CREATE MAP` sentence.

Finally, the sentence `GENERATE GIS` would transform all the specifications made with previous sentences into the source code of a working system. The resulting GIS would provide the users with forms and listings to create, edit, list, and remove any of the entities defined in the data model. The map viewer would also include all the layers, styles, and maps defined. The resulting system may be missing complex functionalities required by the users. It must be noticed that the purpose of the DSL is not to generate a complete system with arbitrarily complex functionalities, but to generate a functional system that can be extended with more complex functions implemented in the general-purpose programming language.

```
GENERATE GIS name;
```

Listing 5: `GENERATE GIS` sentence.

```
CREATE GIS local_administration_manager USING 25829;
USE GIS local_administration_manager;

CREATE ENTITY Municipality (
  id Long IDENTIFIER,
  name String REQUIRED,
  extension MultiPolygon,
```

```
  roads Road RELATIONSHIP(1..1, 0..*),
  offices AdministrativeOffice RELATIONSHIP(1..1, 0..*)
      BIDIRECTIONAL
);

CREATE ENTITY Road (
  id Long IDENTIFIER,
  status String,
  path MultiLine
);

CREATE ENTITY AdministrativeOffice (
  id Long IDENTIFIER,
  status String,
  location Point,
  municipality Municipality RELATIONSHIP MAPPED_BY offices
);

CREATE TILE LAYER base AS "Base Layer" (
  url "https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
);

CREATE GEOJSON LAYER offices AS "Administrative Offices" (
  AdministrativeOffice EDITABLE,
  fillColor #243452,
  strokeColor #eeeee3,
  fillOpacity 0.8,
  strokeOpacity 0.9
);

CREATE BasePolygonStyle name (
  styleLayerDescriptor
    "/home/user/sld/file_polygon_sld.xml"
);

CREATE BaseLineStyle name (
  styleLayerDescriptor "/home/user/sld/file_line_sld.xml"
);

CREATE WMS LAYER defaultOverlay AS "Overlay" (
  Municipality BasePolygonStyle,
  Road BaseLineStyle
);

CREATE SORTABLE MAP map AS "Map Viewer" (
  base IS_BASE_LAYER,
  defaultOverlay,
  offices HIDDEN
);

GENERATE GIS local_administration_manager;
```

Listing 6: Example of application defined by the DSL.

4 USE EXAMPLE

In this section, we present an example of use of our DSL. First we describe an intentionally simplified GIS application. Then we show how we specify this application using our DSL.

Local administrations usually need to manage a set of buildings related with many different areas, such as water distribution buildings (pipes, wells, tanks, chlorination stations, etc.), road networks

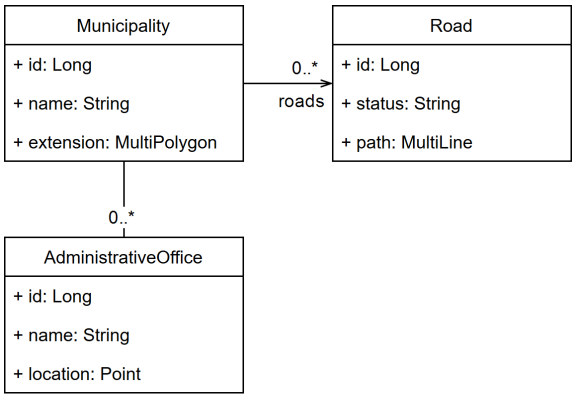

Figure 3: Data model of the example application.

(streets, municipal roads, bridges, etc.), cultural-related buildings (schools, sport halls, community centres, etc.), or administrative related buildings. For a long time now, most of the applications managing this kind of data are based on GIS technologies, allowing the users to visualize the information through map viewers and to digitize new elements.

Using our DSL, we could define such application. Figure 3 shows a simplified data model that specifies that we manage municipalities, roads, and administrative offices, each one with its geographic component (a multi-polygon for municipalities, a multi-line for roads, and a point for administrative offices) and with their relationships. Listing 6 shows the DSL code that specifies the web-based GIS application that manages that data model. First, the new application is created. We use the SRID 25829, which is a local reference system commonly used when working in the north-west of Spain.

Next, we define the data model, creating its three entities. The names of the entities will be used afterward for defining the different layers that will be provided by the application. These layers are also linked to the only map viewer we define, in which it will appear a TileLayer from Open Street Maps (OSM) that works as the base layer, a WMS Layer that combines both the municipalities and the roads, and a GeoJSON Layer with the administration offices. The latter also allows accessing a form directly from the map viewer so the offices can be edited.

Finally, the GIS application could be generated, to produce a software with forms, listings, and maps to manage the entities, layers, and the map we defined.

5 CONCLUSIONS AND FUTURE WORK

GIS is a domain in which most assets are shared among many products, such as features or capabilities, technology, libraries, etc., and the main difference between different applications are changes in the data model. Therefore, it is a suitable domain to apply MDE to facilitate the early stages of development.

In this paper, we have proposed a high-level declarative DSL for GIS specification and development. This language allows the developer to define the different elements that will be managed in the application, and how they are going to be visualized: we can define the entities of the data model, their properties (including spatial data types), and the relationships between them, different types of layers (Tile, WMS, and GeoJSON layers), and maps that will show the data according to the definition of the entities, the layers, and their corresponding styles.

The purpose of the proposed DSL is to allow the developers to automatically generate a GIS application that will include forms, listings, and maps to manage the defined entities and to visualize them according to the specification. In most cases, a real system will surely include complex functionalities that cannot be specified with our language. We consider that those arbitrarily complex functions are out of the scope of our language since the best way of implementing them will be, in most cases, to do that directly in the final project. That is, while our language may not be able to generate a complete GIS for any application, it allows the developer to generate a base system with all the functionalities that can be generated automatically.

Future work includes implementing the interpreter and code generator corresponding to our language and validating its use through experiments in which we try to generate existing applications.

ACKNOWLEDGEMENTS

Partially funded by: Xunta de Galicia/FEDER-UE CSI: ED431G/01 (Centros singulares de investigación Galicia); Xunta de Galicia/FEDER-UE CSI: ED431C 2017/58 (Grupo de Referencia Competitiva); Xunta de Galicia / FEDER-UE, ConectaPeme, GEMA: IN852A 2018/14; MINECO-AEI/FEDER-UE Datos 4.0 (TIN2016-78011-c4-1-R); MINECO-AEI/FEDER-UE ETOME-RDF3D3 (TIN2015-69951-R); MINECO-AEI/FEDER Flatcity (TIN2016-77158-C4-3-R); MICINN-AEI/FEDER-UE BIZDEVOPS: (RTI2018-098309-B-C32).

REFERENCES

Behrens, H. (2010). Mdsd for the iphone: developing a domain-specific language and ide tooling to produce real world applications for mobile devices. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, pages 123–128. ACM.

Bernardino, M., Zorzo, A. F., and Rodrigues, E. M. (2016). Canopus: A domain-specific language for modeling performance testing. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 157–167. IEEE.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182.

Cortiñas, A., Luaces, M. R., Pedreira, O., and Places, Á. S. (2017a). Scaffolding and in-browser generation of web-based gis applications in a spl tool. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume B*, pages 46–49. ACM.

Cortiñas, A., Luaces, M. R., Pedreira, O., Places, Á. S., and Pérez, J. (2017b). 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*, pages 190–194. ACM.

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

Kosar, T., Bohra, S., and Mernik, M. (2016). Domain-specific languages: A systematic mapping study. *Information and Software Technology*, 71:77–91.

Kramer, D., Clark, T., and Oussena, S. (2010). Mobdsl: A domain specific language for multiple mobile platform deployment. In *2010 IEEE International Conference on Networked Embedded Systems for Enterprise Applications*, pages 1–7. IEEE.

Lolong, S. and Kistijantoro, A. I. (2011). Domain specific language (dsl) development for desktop-based database application generator. In *Proceedings of the 2011 International Conference on Electrical Engineering and Informatics*, pages 1–6. IEEE.

Longley, P. A., Goodchild, M. F., Maguire, D. J., and Rhind, D. W. (2015). *Geographic information science and systems*. John Wiley & Sons.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Miranda, M. A., Ribeiro, M. G., Marques-Neto, H. T., and Song, M. A. J. (2017). Domain-specific language for automatic generation of uml models. *IET Software*, 12(2):129–135.

Pastor, O. and Molina, J. C. (2007). *Model-driven architecture in practice: a software production environment based on conceptual modeling*. Springer Science & Business Media.

Pons, C. F., Giandini, R. S., and Pérez, G. A. (2010). Desarrollo de software dirigido por modelos.

Ribeiro, A. and da Silva, A. R. (2014). Xis-mobile: A dsl for mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, pages 1316–1323. ACM.

Rosique, F., Losilla, F., and Pastor, J. Á. (2017). A domain specific language for smart cities. In *Multidisciplinary Digital Publishing Institute Proceedings*, volume 2, page 148.

Salihbegovic, A., Eterovic, T., Kaljic, E., and Ribic, S. (2015). Design of a domain specific language and ide for internet of things applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 996–1001. IEEE.

Worboys, M. F. and Duckham, M. (2004). *GIS: a computing perspective*. CRC press.