International Journal of Geo-Information

MDPI

*Article*

# A Geospatial Application Framework for Directional Relations

**Eliseo Clementini* and Giampaolo Bellizzi**

Department of Industrial and Information Engineering and Economics, University of L'Aquila, 67100 L'Aquila, Italy; gbelliz84@gmail.com

* Correspondence: eliseo.clementini@univaq.it

Received: 6 December 2018; Accepted: 13 January 2019; Published: 15 January 2019

**Abstract:** Geographic data analysis is based on the use of spatial relations as a means of selecting and processing geometric data associated with geographic features. Starting from 1990, topological relations have been recognized as fundamental criteria in geographic data processing, leaving out other kinds of spatial relations, such as directional relations. The latter ones, despite having quite an important role in geospatial applications, have been developed as theoretical models but very little implemented in systems. We refer in this paper to the 5-intersection model for expressing projective relations that can be used to implement directional relations in various frames of reference. We design an application framework in Java and use the framework for answering various categories of queries involving directions. We finally outline how to use the framework for validating the cognitive adequacy of relations with user tests.

**Keywords:** Directional relations; Frames of reference; Geographic query processing

## 1. Introduction

The significant increase in the availability of georeferenced data has resulted in the development of many Geographical Information Systems (GIS) applications in recent years. Reference open standards in the spatial data panorama are established by several organizations, such as the Open Geospatial Consortium (OGC), the International Organization for Standardization (ISO), and the International Hydrographic Organization (IHO). From the onset, the definition of appropriate spatial relations that eased the man-machine interaction towards georeferenced data seemed to be one of the main objectives on which research should concentrate its work. The first spatial relations to be developed were topological ones [1–3]. More in general, spatial relations have been categorized in topological, projective, and metric relations [4,5].

Projective geometry can be used to build a comprehensive model that covers intermediate relations between topology and metrics. To have a qualitative understanding of projective relations, it is helpful to think about different two-dimensional views of a three-dimensional real-world scene of objects: changing the point of view, metric aspects such as distances and angles among the objects appear to be different, but there are properties that are common in all the views. These common properties are projective properties. In this context, work on ternary projective relations (5-intersection model) was proposed in the literature [6–9]. Projective relations include common subcategories such as directional relations and visibility relations [10–12].

Directional relations represent a category of spatial relations that provide information about the spatial arrangement of objects in a scene or locations in an embedding space [13,14]. Directional relations are in most cases binary relations between two spatial objects, but a frame of reference is needed to disambiguate the meaning of the relation in a given context [15–17]. Frames of reference in

connection with directional relations have been studied since long time from historical classifications [18–20] to recent developments [21–23].

In [21], the author defines a model of directional relations in various frames of reference, specifying the main types and subtypes: this outcome can be considered an ontology of frames of reference, refining the classification of [19]. The knowledge of directions and frames of reference allows the definition of a set of directional operators that can be integrated in a spatial query language. The directional relations are mapped to the 5-intersection model, which precisely gives the geometric definitions and the algorithms to compute relations.

In this paper, we develop an application framework in Java that implements the model of [21]. This implementation corresponds to a library capable of managing the various frames of reference and their directional relations. We evaluate the application framework by running user tests on a small data set of an urban area and we outline how the framework can be the basis for testing the cognitive adequacy of the implemented directional operators.

The remainder of this paper is organized as follows. In Section 2, we summarize background information on the 5-intersection model and on directional relations. In Section 3, we describe the requirements of the application framework and its general architecture. In Section 4, we describe use cases to evaluate the application framework. In Section 5, we discuss user tests of the application framework. Section 6 gives short conclusions.

## 2. Background Information

### *2.1. The 5-Intersection Model*

A ternary projective relation expresses the position of a primary object $A$ with respect to a frame of reference made up by two other regions $B$ and $C$. In the 5-intersection model, the frame of reference is built around the two reference regions $B$ and $C$ as shown in Figure 1. Region $B$ is also called first reference object ($RO_1$) and region $C$ is called second reference object ($RO_2$). There is a direction from $RO_1$ to $RO_2$ that is necessary to determine the name of the areas. Hence, the five areas of the construction are called *Before*($B$,$C$), *Between*($B$,$C$), *After*($B$,$C$), *Leftside*($B$,$C$), and *Rightside*($B$,$C$). Let us consider the following matrix of empty/nonempty intersections of a region $A$ with such five areas:

| | $A \cap Leftside(B,C)$ | |
|---|---|---|
| $A \cap Before(B,C)$ | $A \cap Between(B,C)$ | $A \cap After(B,C)$ |
| | $A \cap Rightside(B,C)$ | |

The previous matrix is called the *5-intersection*. In the matrix, a value 0 indicates an empty intersection, while a value 1 indicates a nonempty intersection. Since the matrix with all five values equal to zero does not correspond to a relation, the 5-intersection model is able to identify 31 projective relations. Due to space limitations, we show only some of them in Figure 2. The five basic relations where the primary region $A$ falls entirely inside one area are called *rightside*, *leftside*, *between*, *before*, and *after*, respectively. When the primary region $A$ falls in more than one area, there are compound relations such as *before:rightside*. More details on the geometric construction and on the properties of the model can be found in [24].

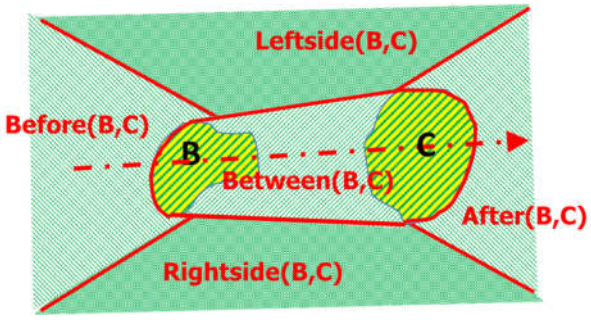



**Figure 1.** The partition of the plane into five areas by reference regions *B* and *C* (figure taken from [25]).

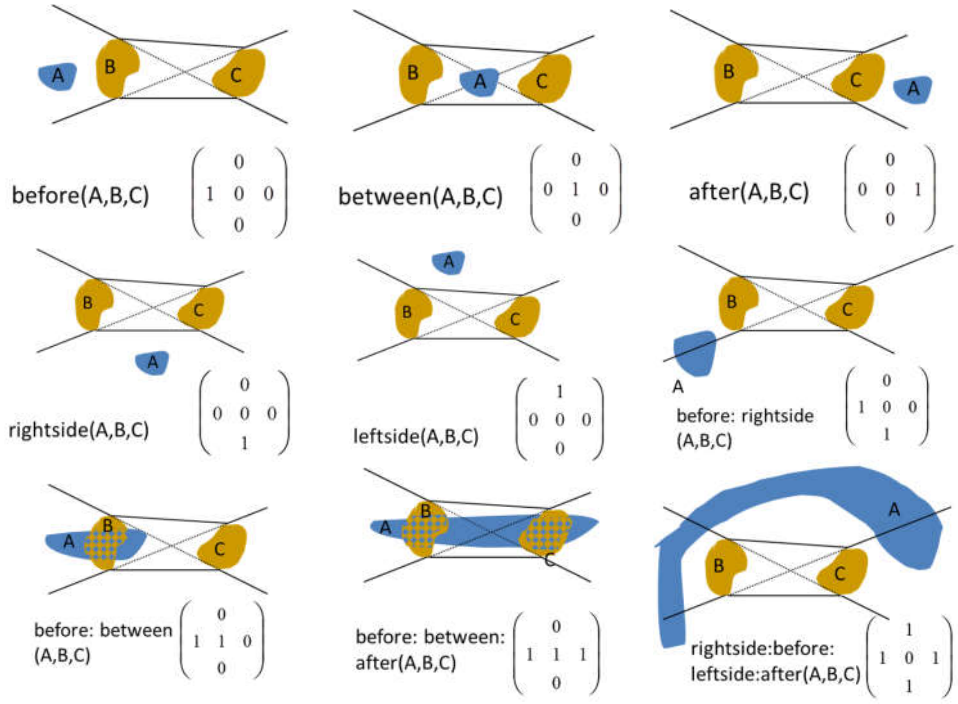



**Figure 2.** Some representative relations of the 5-intersection model (figure partly taken from [25]).

## 2.2. *Frames of Reference*

In Figure 3, we show a categorization of frames of reference as proposed in [21]. Hereafter we restrict the description to three of them for space limitation: the extrinsic frame of reference of subtype 'geographic—conventional North' (abbreviated in the rest of the paper as the 'geographic' frame of reference), the intrinsic frame of reference of subtype 'geometric' (abbreviated in the rest of the paper as the 'geometric' frame of reference), and the deictic frame of reference of subtype 'allocentric' (abbreviated in the rest of the paper as the 'deictic' frame of reference). These three subtypes of frames of reference will be used throughout the paper as a recurrent case study. For an explanation of all frames of reference, we point to the original paper [21].

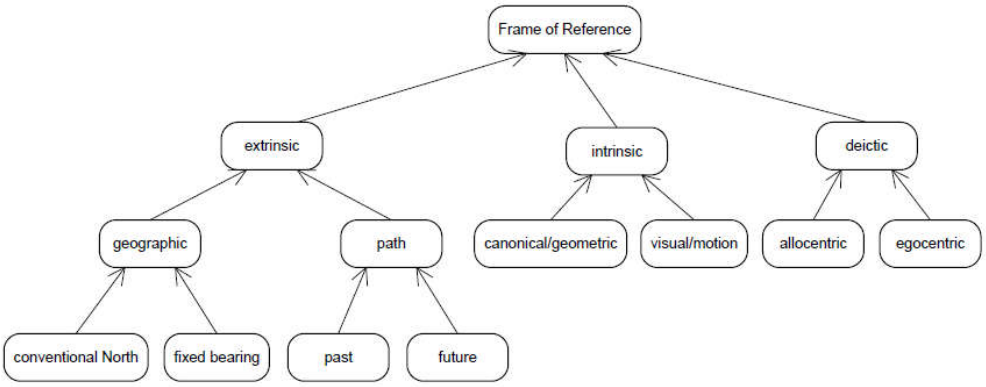



**Figure 3.** Taxonomy of frames of reference (figure taken from [21]).

In the geographic frame of reference, reference objects *B* and *C* of the 5-intersection model correspond to the first reference object and the conventional North line as illustrated in Figure 4. In this frame of reference, the directional relations *North_of*, *East_of*, *South_of*, and *West_of* are identified. They can be mapped to 5-intersection relations by the following transformation function for the extrinsic-geographic-conventional North (EGN) frame of reference $\Phi_{EGN}$:

| $\Phi_{EGN}$ | |
|---|---|
| $r$ | $r'$ |
| *North_of* | *between* |
| *East_of* | *rightside* |
| *South_of* | *before* |
| *West_of* | *leftside* |
| | *after* |


**Figure 4.** Geographic frame of reference.

We can notice that by shortening or enlarging the conventional North line, the extent of the areas assigned to cardinal directions is modified accordingly. Also, based on the projection, the tangent lines to objects might not be straight. Therefore, the second reference object in the geographic frame of reference is an important parameter to be chosen. As an example of query, if the user is interested to find the countries that are south of Algeria, the corresponding SQL query would be:

```
SELECT c1.name FROM country AS c1, country AS c2
WHERE c2.name = "Algeria" AND south_of(c1.the_geom, c2.the_geom);
```

As a second subtype, we discuss now the geometric frame of reference. In this case, objects *B* and *C* of the 5-intersection model correspond to 'back' and 'front' sides of the reference object in Figure 5. In this frame of reference, the directional relations *inside, rightside, back, leftside,* and *front* are identified. They can be mapped to 5-intersection relations by the following transformation function for the intrinsic-geometric (IC) frame of reference $\Phi_{IC}$:

| $\Phi_{IC}$ | |
|---|---|
| *r* | *r'* |
| *inside* | *between* |
| *rightside* | *rightside* |
| *back* | *before* |
| *leftside* | *leftside* |
| *front* | *after* |

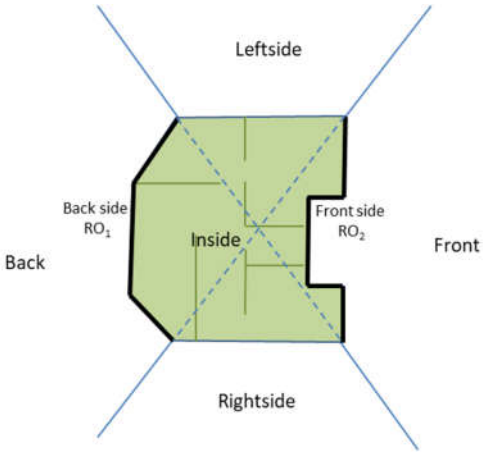



**Figure 5.** Geometric frame of reference.

An example of query for the latter frame of reference can be illustrated as follows (see Figure 6). If the user wants to know: "What are the supermarkets in front of the building where I live?", then, combined with some distance information (not the goal of this paper), the SQL query becomes:

```
SELECT b1.id FROM building AS b1, building AS b2
WHERE b2.name = "my house" AND front(b1.the_geom,b2.the_geom) AND
near(b1.the_geom, b2.the_geom);
```

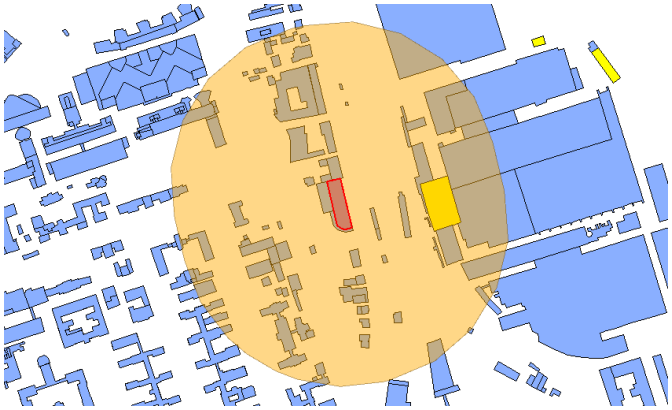


**Figure 6.** Finding the supermarkets in front of a given building and at a short distance.

The last frame of reference that we discuss is the deictic one. In this case, objects *B* and *C* of the 5-intersection model correspond to the observer position and the reference object in Figure 7. In this frame of reference, the directional relations *front, right_of, left_of,* and *behind* are identified. They can be mapped to 5-intersection relations by the following transformation function for the deictic-allocentric (DA) frame of reference $\Phi_{DA}$ :

| $\Phi_{DA}$ | |
|---|---|
| *r* | *r′* |
| *front* | *between* |
| *right_of* | *rightside* |
| | |
| *left_of* | *leftside* |
| *behind* | *after* |

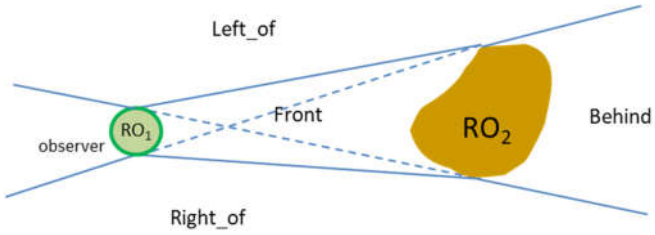



**Figure 7.** Deictic frame of reference.

An example of deictic frame of reference can be illustrated as follows (see Figure 8). Let us suppose that a tourist is walking in an urban context and looking toward a monument he(she) asks what the buildings are on his/her left side. Then the corresponding SQL query would be:

```
SELECT a.id FROM building AS b, building AS a
WHERE b.name = "monument" AND leftside(a.the_geom,b.the_geom);
```

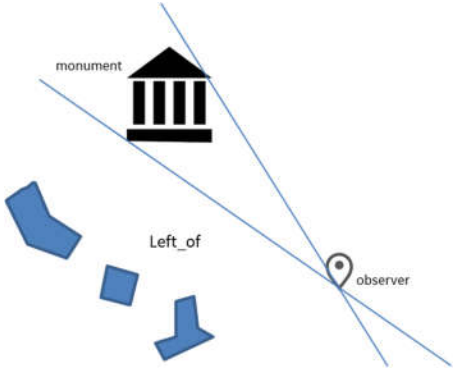



**Figure 8.** Finding buildings on the left side of the monument from the observer point of view.

## 3. Application Framework

In this section, we illustrate the requirements that the application framework for directional relations should satisfy and the overall design of the architecture, which is based on three layers and is implemented in the Java programming language.

### *3.1. Requirements*

The framework for directional relations is based on the 5-intersection model: therefore, an essential component of the architecture needs to be a software library (e.g., in Java) capable of calculating the projective relations of such a model. Then, the application framework needs to represent various frames of reference. Each of them needs to have a corresponding class in which the methods to calculate the directional relations associated to it are included. The directional relations of each class are calculated with the corresponding mapping to the projective relations library. At the top of the class hierarchy, we assume an abstract class *FoR* (*Frame of Reference*), from which other abstract classes will inherit, one for each type of frame of reference. The instantiable classes are instead at the lower hierarchical level, for which a series of information should be defined.

The abstract class *FoR* needs to include the following abstract methods, which will be implemented in derived classes:

- The method *Fi* that implements the function Φ described in Section 2.2, which translates a directional relation into a 5-intersection relation.
- A first method *relate* able to check whether a directional relation between two objects is verified or not. Possibly, also multiple relations could be verified (e.g., *behind:left_of*). For example, a query could have the form: "Is the tree on the left or behind the university building?"
  ```
  boolean test_rel = fr_of_ref.relate(tree, university, after:left_of)
  ```
  Alternatively, the relation could also be specified with a binary pattern using appropriate methods *patt2rel* and *rel2patt,* which would guarantee a translation from a binary string to the relation and vice versa. For example:
  ```
  boolean test_rel = fr_of_ref.relate(tree, university, "00011")
  ```
- A second method *relate* able to return all objects that are in some relation with a given object. This function can be used to search for objects from a data store, filtering them so that they verify the relation expressed in the query.

For each type of frame of reference, all subtypes need to be implemented with concrete classes. For example, Figure 9 shows The UML diagram for the deictic-allocentric frame of reference $f_{DA}$. Every class representing a subtype will have the same structure, including the following attributes and methods:

- the *Subtype* of the frame of reference;
- a geometric object *Object*, which is of different meaning depending on the frame of reference (basically, the third object of the ternary relation that is not explicit in the binary relation);
- a set *Rs* of all possible relations that can be used with the frame of reference. Each relation can be completely defined in terms of a pattern: *Rs* is an array of two-dimensional strings containing the relation name and its pattern. For example, for the deictic-allocentric frame of reference, *Rs* will be the following:
  ```
  Rs = new String [5][2];
  Rs[0][0] = "10000";
  Rs[0][1] = "front";
  Rs[1][0] = "01000";
  Rs[1][1] = "right_of";
  Rs[3][0] = "00010";
  Rs[3][1] = "left_of";
  Rs[4][0] = "00001";
  Rs[4][1] = "behind";
  ```
  Note that in this frame of reference, one relation of the 5-intersection model (corresponding to the *before* relation (or "00100") is not used.
- The transformation function *Fi,* which for the deictic-allocentric frame of reference corresponds to the function $\Phi_{DA}$ described in Section 2.2;
- A set of Boolean functions that can compute the individual relations defined on the frame of reference. These relations are binary, i.e., they operate between two objects by accessing their

geometric representation. Let us consider the following example: "Is the tree to the left of the university, with respect to the viewpoint of an observer?". To verify this type of request, the function to be invoked is *left_of*, which is a method of the class *FoR_Deictic_Allocentric*. The call to this method is as follows:

```
boolean test_rel = fr_of_ref.left_of(tree,university)
```

We assume that spatial objects used in the queries belong to the class *org.postgis.Geometry*, which represents the generic geometry, from which other classes inherit, such as Point, Polygon, and Polyline, according to OGC standard.

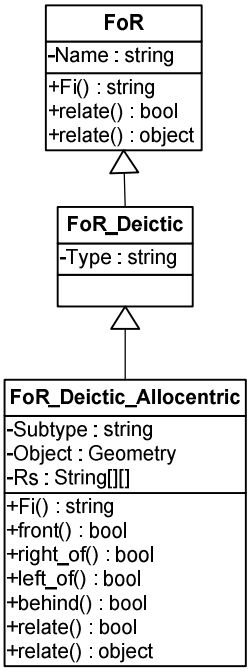



**Figure 9.** The UML diagram of the deictic-allocentric frame of reference.

### *3.2. Architecture of the Framework*

A layered architecture can guarantee a high level of interoperability among components. In *data-centric* applications, the decomposition process allows us to identify at least three main logical levels: the presentation layer, the business logic layer, and the data layer. Figure 10 is an overall view of the designed architecture.

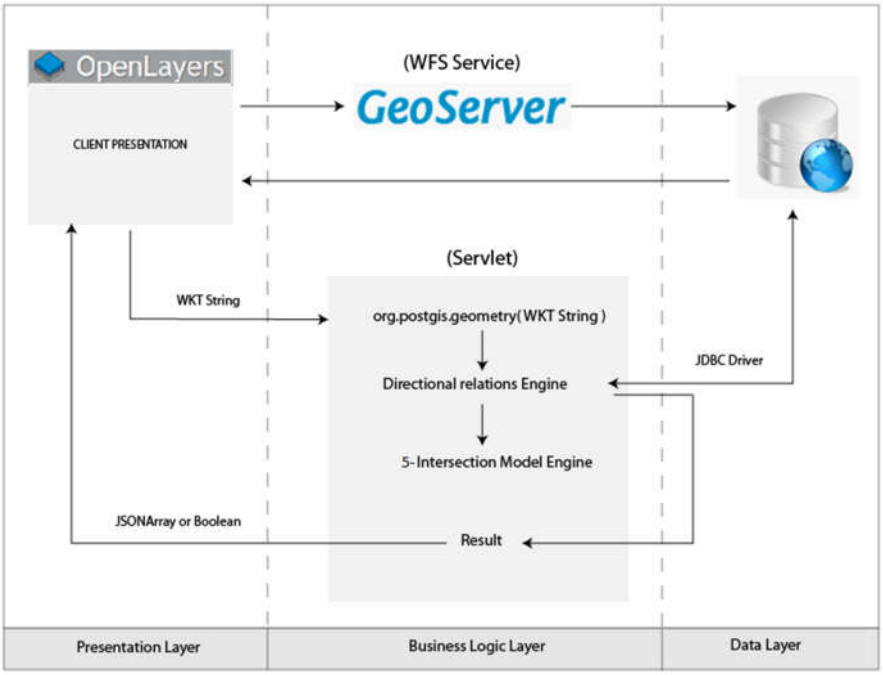



**Figure 10.** Architecture of the application framework.

3.2.1. The Presentation Layer

The presentation layer handles communications with entities external to the system. The layer includes all the components that deal with the presentation of information to the client, allowing a high-level interaction with the user. As a presentation layer, our implementation uses the framework Openlayers, which is a JavaScript library for displaying maps within a web browser, released under the 2-clause BSD license. OpenLayers allows the creation of real GIS applications in web. The presentation layer communicates with the business logic layer via a web service. There are many web services available in the OGC standard for exchanging geospatial information. We are particularly interested in vector data and, therefore, the web service to be used is the Web Feature Service (WFS) that allows the exchange of vector data in the OGC standard. Let us deepen how OpenLayers works:

```
var protocol = new OpenLayers.Protocol.WFS({
                        version: "1.1.0",
                        srsName: "EPSG:4326",
                        url: "http://localhost:8080/geoserver/wfs",
                        featureType: "feature_name",
                        geometryName: "the_geom"
})
```

With the above code, OpenlLayers permits to declare a WFS protocol with which it is possible to send all the requests to the server. New features can be created, and existing ones can be retrieved, updated, or deleted. The parameters in the above code specify several details, such as the version of the WFS protocol, the map projection being used (*srsName*), the web address of the server (*url*), the type of geometric objects that are requested to the server (*featureType*), and the name of the geometric attribute (*geometryName*). After the creation of the protocol, it is necessary to create a vector layer to visualize the required features. Layers can be overlaid to relate different datasets (such as rivers and lakes). To create a vector layer, we write:

```
var vecLayer =
new OpenLayers.Layer.Vector("Layer_name",{isBaseLayer: true});
```

Besides specifying the layer name, the parameter *isBaseLayer* is used to indicate whether it is the main layer or not. Next, it is necessary to bind the newly created layer to the previously defined WFS protocol. In Java this will correspond to the following code:

```
var store = new GeoExt.data.FeatureStore({
            layer: vecLayer,
            fields: [{name: 'tipo', type: 'string'}],
            proxy: new GeoExt.data.ProtocolProxy({
                protocol: protocol
            }),
            autoLoad: true
});
```

Specifically, in the last piece of code, we invoked the library *GeoExt*, which includes the library Openlayers. The parameters used for binding are the name of the previously defined vector layer, additional attributes (*fields*), the previously defined WFS protocol (*proxy*), and a Boolean parameter that specifies whether the data should be automatically loaded on the layer (*autoLoad*).

In Openlayers, the user can select some features in the vector layer and send them to the server. The selection of features is managed through events. Technically, this is done with the following statement:

```
vecLayer.events.register("featureselected", vecLayer, onFeatureSelect);
```

The event management is directly enabled on the created layer. When the event *featureselected* occurs, the function *onFeatureSelect* is invoked. The latter will accomplish the task of sending the selected geometries to the server through an AJAX request (Asynchronous JavaScript and XML), which has the following structure:

```
$.ajax({
   type: "POST",
   url: "http://localhost:8080/pattern/App_Name",
   data: data,
   //event of success
   success: function(msg){
     if (!bool){
       Ext.getCmp('log').update('resulting features are shown in red..');
       var json = new OpenLayers.Format.JSON();
       var res = json.read(msg);
       handler(res);
     }else {
       Ext.getCmp('log').update('Relation result: '+msg);
     }
   }
});
```

The above request is handled through *Jquery*, a JavaScript framework able to manage several AJAX features. The parameters that we can see in the request are the type, which is set to *POST* because the features go from the presentation layer to the server (otherwise, the type would be set to *GET*), the web address to which the request should be sent (*url*), the data to be sent to the server, and a function to be invoked if the request has been successfully completed. In the latter function, the parameter *msg* represents the value that the server provides in response to the request.

#### 3.2.2. The Business Logic Layer

The business logic layer handles the logic of the application. In our framework, for example, this layer will have to manage both the WFS request coming from the presentation layer to display the

features of the selected geographic region and the AJAX request coming as well from the presentation layer to instantiate the right classes to be able to use various directional relations.

To deal with a WFS request, a WFS server is needed. We chose GeoServer, an OGC standard compliant server, multiplatform and written in Java. GeoServer needs a web container, for which we chose Apache Tomcat. The next step is connecting GeoServer to the spatial database of a given geographic region. A new store needs to be created by accessing to the section *stores* of GeoServer's main page. At this step, a source for the store must be indicated, such as the PostGIS database entry in the vector data sources section. For the connection, some parameters are required (e.g., host, port, database name). After the store, a layer must be created that will then be accessible through any WFS request. Figure 11 shows the layer's publishing phase. Among the parameters to be specified are the name to be assigned to the layer, the coordinate reference system of the data set, and the coordinates of the minimum bounding rectangle (MBR) containing all the features of the layer.

Once some geometric features are selected on the vector layer of the client, the geometric data is transformed by OpenLayers to strings in the Well-Known Text (WKT) format. These strings are then sent to the server for using them in the calculation of directional relations. The following line shows an example of how the servlet on the business logic layer manages to retrieve this data from the client:

```
String aStr = request.getParameter("a");
```

The geometries are transformed into objects of the class *org.postgis.Geometry*. This can be easily achieved because the constructor of the class allows the instantiation of objects from a string in the WKT format. For example, to create a polygon from the previously created string a*Str*, we need to use the following statement:

```
org.postgis.Polygon a = new org.postgis.Polygon(aStr);
```

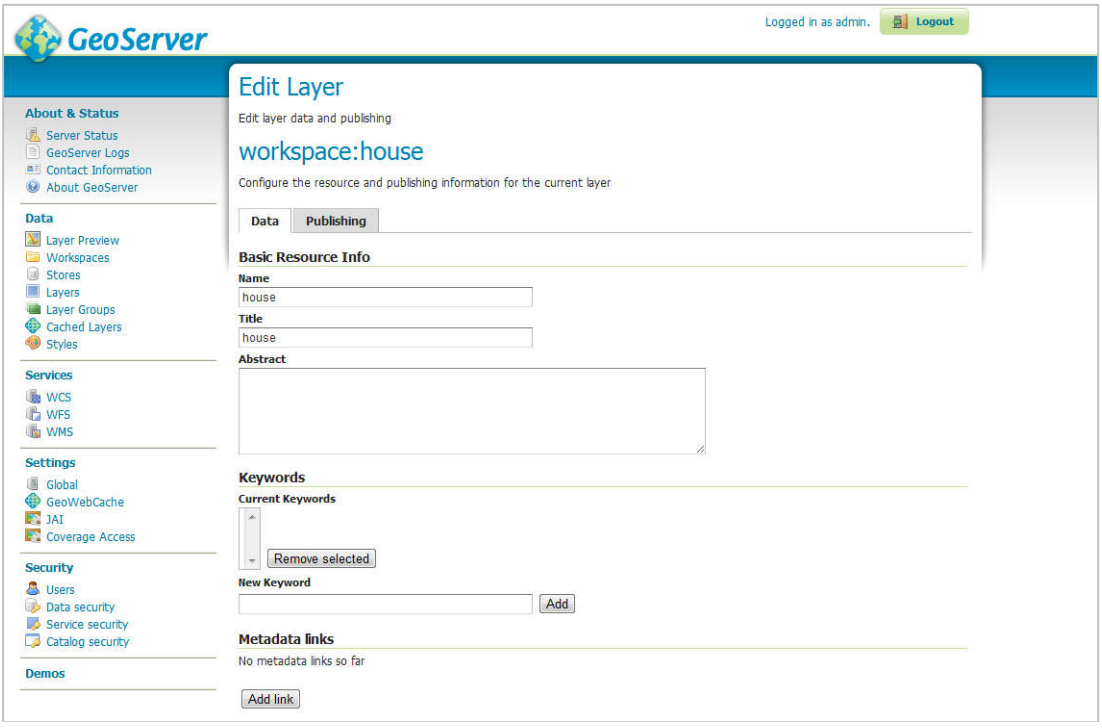



**Figure 11.** Publishing a layer.

Then, it is necessary to instantiate the class for the required directional relation. Supposing that the user has chosen the geographic frame of reference and requires checking if an object A is north of an object B, we first need to instantiate the frame of reference with the following code:

```
FoR_Extrinsic_geographic_conventional_north geo_fr =
```

```
new FoR_Extrinsic_geographic_conventional_north ("Test", "Extrinsic",
"Geographic", "conventional_north", North.getNorth(cStr));
```

The class creates the object representing the geographic extrinsic frame of reference, where the necessary parameters are specified: in detail the last parameter is the geometry of the north line, returned by the function *getNorth* of the North class (a simple utility class). To calculate if the directional relation between A and B is the "north", it is possible to apply the method *relate* in the following way:

```
Boolean ris = geo_fr.relate(a, b, north_of);
```

Instead, if the user requests all objects to the north of a selected object B, the method to be invoked will be the non-Boolean *relate* that returns a collection of objects. The following call will be used:

```
ris = geo_fr.relate(list, b, north_of);
```

The parameter *list* contains all the objects in the dataset for which the requested relation is verified. The resulting objects are then placed in a JSON array that will be returned to the client. Hence, the parameter *list* must be able to access the spatial database: the responsible Java class is located in the data layer.

#### 3.2.3. The Data Layer

The *Data Layer* is the part of the architecture that actually performs data access. In our application framework, it is the class *DbData* that executes this task by using the static method *getDbData*. Therefore, the following method is able to receive the request from the business logic layer, transform it in a SQL query, send such a query to the spatial database, extract the results and sending them back to the business logic layer as a list of spatial objects. Technically, we can observe the code of this method as follows:

```
static public ArrayList <org.postgis.Geometry> getDbData(String
exclude, String dbName,String tbName, String user, String pass) throws
ClassNotFoundException, SQLException{

     Class.forName("org.postgresql.Driver");
     String url = "jdbc:postgresql://host:5432/"+dbName;
     Connection conn = DriverManager.getConnection(url, user, pass);

     PreparedStatement ps = conn.prepareStatement("SELECT the_geom
     FROM \""+tbName+"\" where the_geom not in (select
     ST_GeomFromText(?,4326))");

     ps.setString(1,exclude);
     ResultSet res = ps.executeQuery();

     ArrayList<org.postgis.Geometry> list=
     new ArrayList<org.postgis.Geometry>();

     while (res.next()) {

             PGgeometry geometry = (PGgeometry)res.getObject(1);
             list.add(geometry.getGeometry());

     }
ps.close();
conn.close();
return list;}
```

The first task of the above method is to register the driver needed to connect with the database with *Class.forName*; at this point the *DriverManager* will be aware of this driver. The *url* parameter specifies where the database is located; following that, the connection can be created with the method *DriverManager.getConnection*, to which three parameters are passed on: *url*, *user*, and *password* for the logon credentials. Right after, a *PreparedStatement* allows to submitting SQL commands with one or more parameters. In this way, the same SQL command can be invoked several times with different values, where the parameters are indicated with a question mark. The query is then processed by the method *executeQuery* and inserted in the *ResultSet res*. Finally, through the while loop, all query results will be placed in the array *list* that will be returned after closing the connection.

## 4. Evaluation of the Application Framework

To evaluate the application framework, we have run tests by using a dataset of buildings in a suburban area of the town of L'Aquila (Italy): specifically, a vector layer in shapefile format offered to us by the municipality, representing geometries of type *Polygon* and projected into the coordinate reference system *EPSG:4326*. We considered the following possible categories of requests to test directional relations:

- *Boolean Geographic*: with this request it is possible to check whether a house is located to the *north, south, west, east* of another house. The result is a Boolean value.
- *Collection Geographic*: it determines all the houses that are located to the *north, south, west, east* of a given house. The result is a collection of objects.
- *Boolean Geometric*: by activating a filter, it is possible to visualize the buildings on which the front and the rear side have been defined. Hence, it is possible to know whether a building is at the *front*, *right*, *left*, or *back* of another building. The return data is a Boolean value.
- *Collection Geometric*: It permits the identification of all the buildings that are at the *front*, *right*, *left*, and *back* of another building.
- *Boolean Deictic*: based on an individual's position, it is possible to check whether a house is to the *left*, *right*, *front*, or *behind* of another house with respect to the considered point of view.
- *Collection Deictic*: like Boolean deictic, but it retrieves all the houses that satisfy the properties.

The main window of the application is shown in Figure 12. On the left side, a frame displays the geometric features of the building of a selected area. Within the frame, we find tools for navigating and zooming the map. In the central part, some descriptive attributes are shown, which identify the type of building (home, shop, school), where each attribute refers to a geometric component. In the right side, there is a form that manages the interaction with the user, where it is possible to specify the directional relation to be used in the query. At the bottom, there is the log panel.

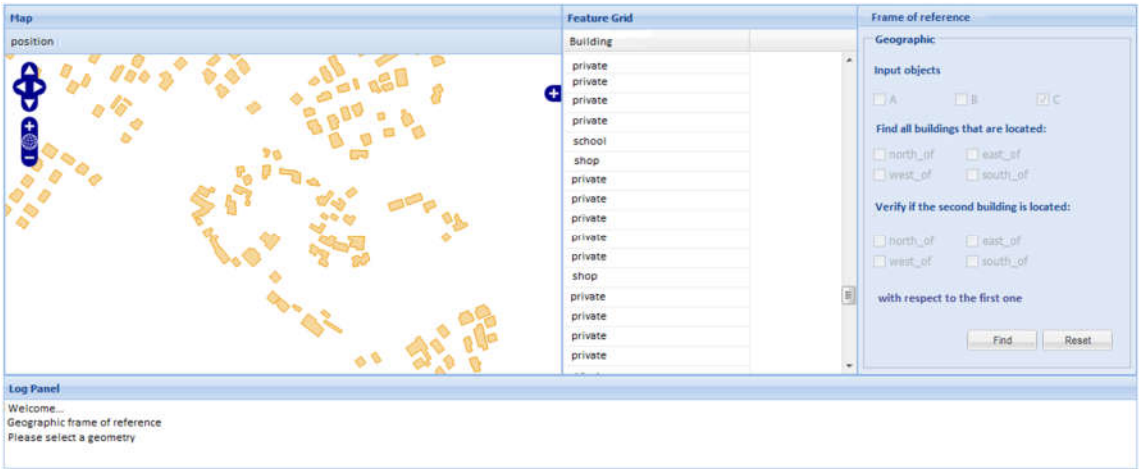



**Figure 12.** The main window of the application.

### *4.1. Geographic Frame of Reference*

In the geographic frame of reference, by selecting one building it is possible to specify a directional relation and return all buildings that are in the given direction with respect to the selected one. Figure 13 shows an example where the relation *north_of* is checked and then the operation was confirmed with the *Find* button at the bottom. It is possible to select multiple directional relations as well. If a second geometry is selected, this will correspond to find the Boolean relation between the two selected buildings (Figure 14).

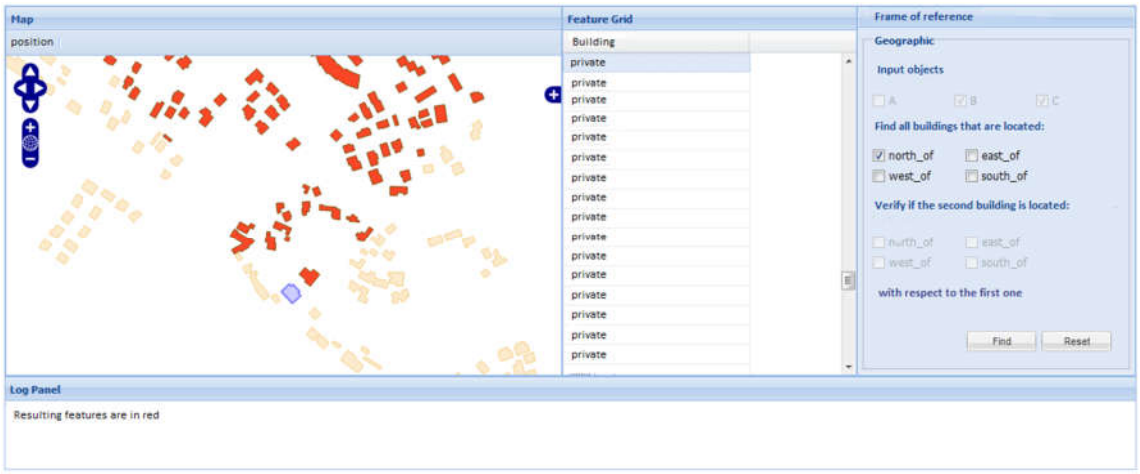



**Figure 13.** Buildings north of the selected one.

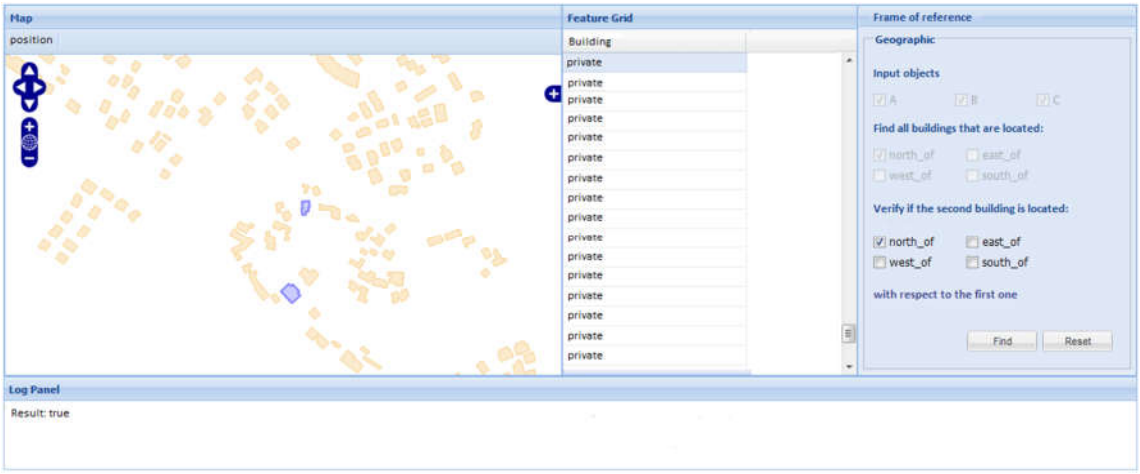



**Figure 14.** Boolean directional relation *north_of*.

### *4.2. Geometric Frame of Reference*

The geometric frame of reference relies on the identification of intrinsic characteristics of the reference object. In the case of buildings, such characteristics are normally represented by the front and back side. In the case study developed here, only a few buildings have this level of detail. On the map, there is a filter that allows the display of these special buildings (Figure 15).

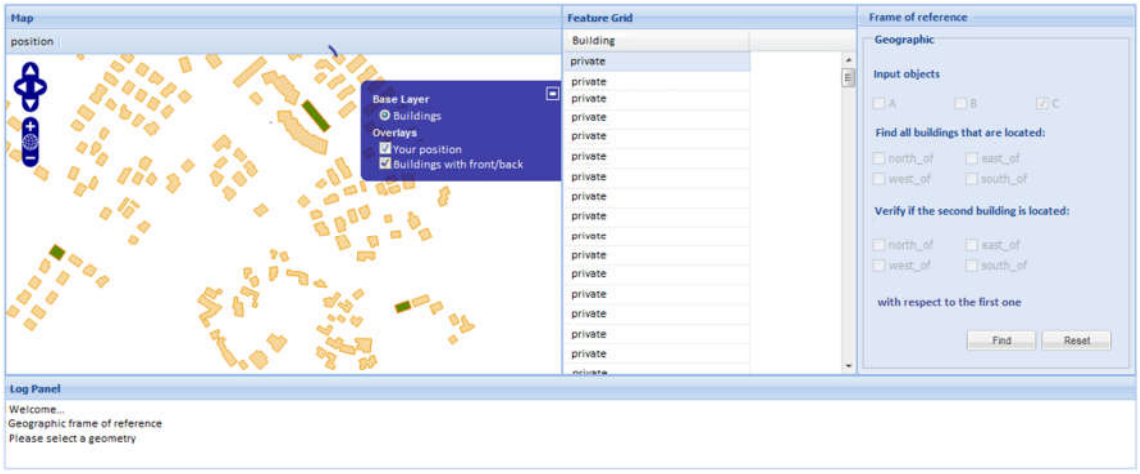



**Figure 15.** Buildings where the front side can be identified.

Such buildings are colored in green: clicking on one of them, the frame of reference will change to geometric. This action will also change the kind of directional relations that can be used with this frame of reference. Figure 16 shows the case in which all the buildings in front of the selected one are requested.

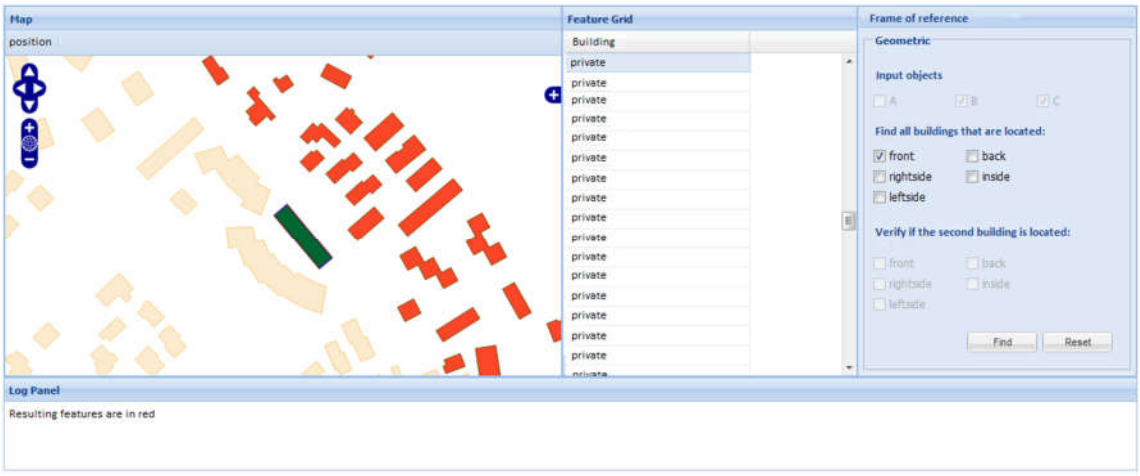



**Figure 16.** Buildings in front of the selected building.

### *4.3. Deictic Frame of Reference*

In this frame of reference, the user is given the possibility to set his/her own position. A specific button *Location* is used for this purpose; when user position is selected, the frame of reference will shift to deictic (Figure 17). As an example, let us assume that the user requires to select all the buildings that are between his/her position and a given building. The user needs to select the latter building, choose the relation *front*, and click on *Find*. In Figure 18, the result is shown.

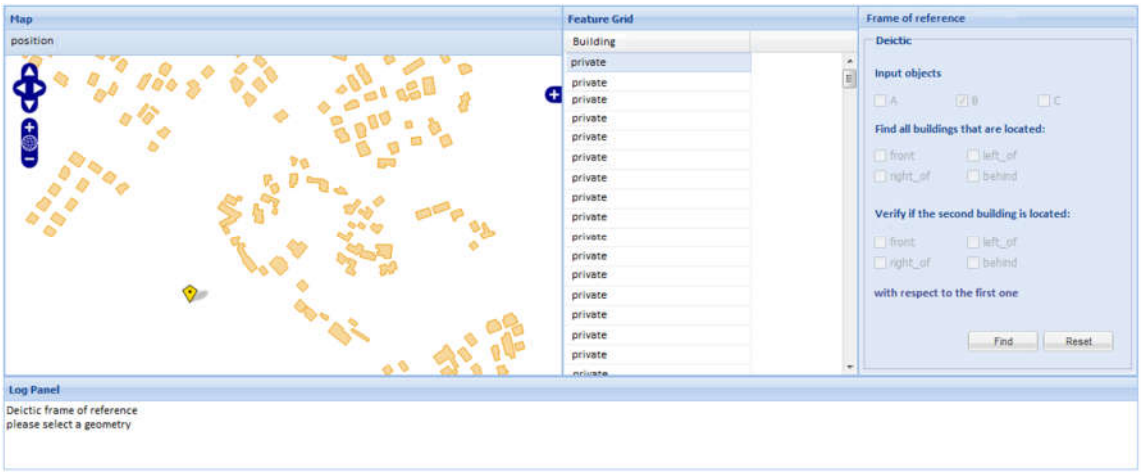



**Figure 17.** Setting user's position in the deictic frame of reference.

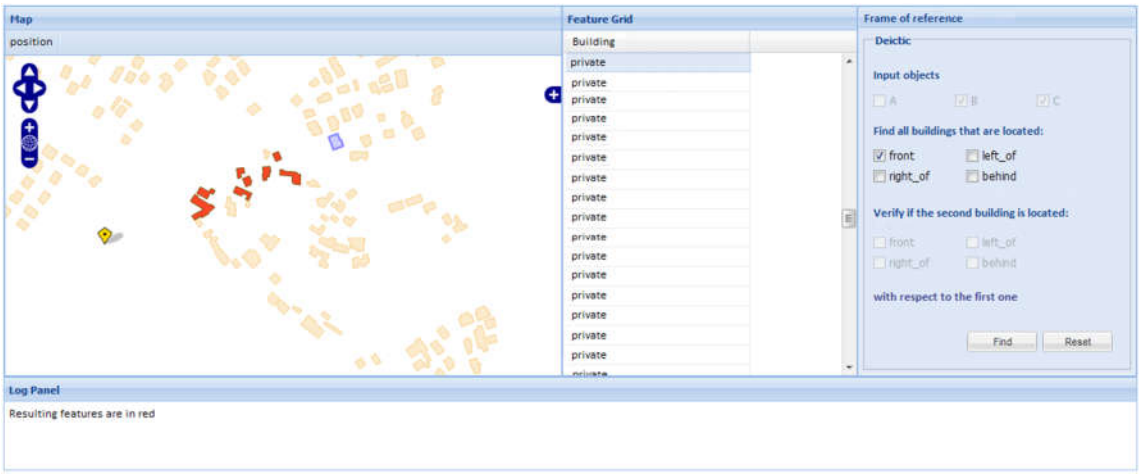



**Figure 18.** Houses in front of the selected building from the observer's point of view.

## 5. User Tests of the Application Framework

In this section, we outline the use of the application framework for testing the cognitive adequacy of the directional relations and frames of reference introduced in [21]. A more complete user study will be the subject of future research.

### *5.1. Preparation of the Test*

The test was prepared by choosing three prototypical subtypes of frames of reference applied to a scene of buildings. We selected 10 participants among the population of students in our university, balanced on gender, and without specific knowledge on mapping and spatial relations. This choice avoided a bias on previous opinions about geographic directions. The participants were confronted with 6 printed scenarios and the question asked to color in red the buildings that in their opinion satisfied the given directional relation with respect to the highlighted building (in yellow). For each of the three subtypes of frame of reference, two different directional relations were tested.

Figure 19a shows the first map examined by participants to answer the first two questions. The map referred to the extrinsic frame of reference of geographic subtype and the questions were to color all the buildings that are north of the highlighted building and west of it, respectively. The third and fourth questions concerned the deictic frame of reference and were based on Figure 19b. In the first question, the user needed to color all the buildings placed between his/her position and the highlighted building. In the second question, the user was asked to color all the buildings located to

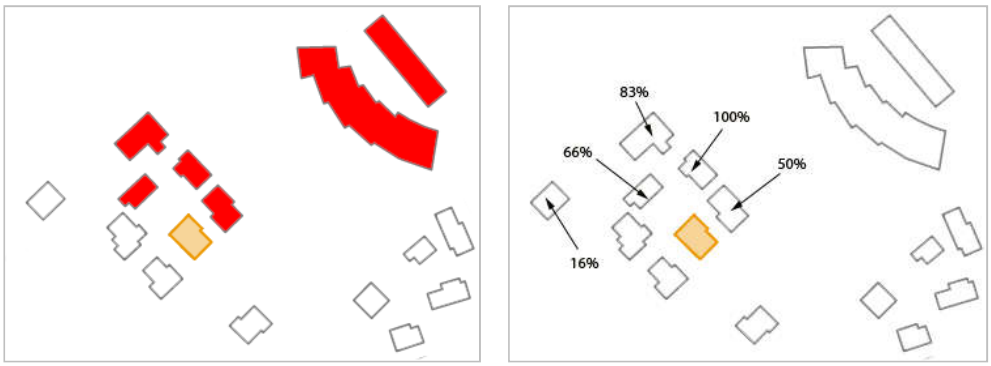
the right of the highlighted building from the point of view of the user's position. The fifth and sixth questions are related to the geometric intrinsic frame of reference. Figure 19c shows the scenario given to the users for tests. Participants were asked to color all the buildings in front of the highlighted building and, in the last question, to color all the buildings to the right of the highlighted building.

(**a**) (**b**) (**c**)

**Figure 19**. Test for the geographic (**a**), deictic (**b**), and geometric (**c**) frame of reference.

*5.2. Test Results*

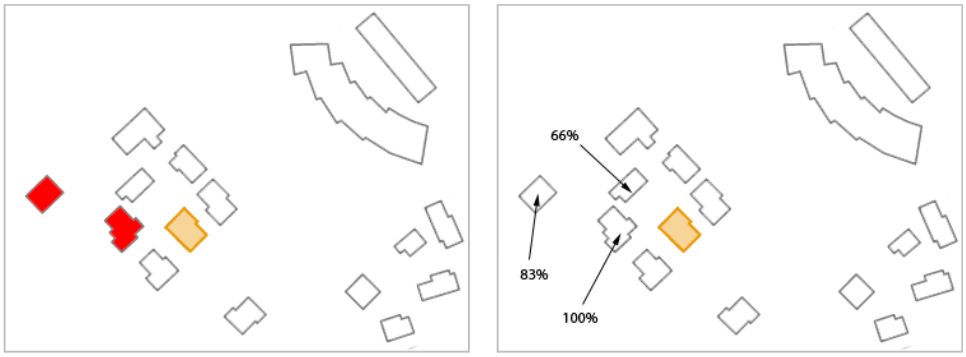
The user tests were evaluated by counting the number of colored buildings in each answer and assigning a percentage of users that selected each building. Then, we compared the user tests with the results coming out from running the application framework. For the geographic frame of reference, the results are shown in Figure 20. In Figure 20a, there is the result given by the application and, in Figure 20b, there is the result obtained from the user tests.




(**a**) (**b**)

**Figure 20**. Results of the *north_of* relation with respect to the selected building: (**a**) from the application; (**b**) from the user tests.

The application results reflect positively the idea that users have of the *north_of* relation. However, there are some buildings that users have not considered and are in the top right corner of Figure 20a. This can be explained by the fact that the application also returns the buildings that are only partly located to the north, but that can be partly east or west as well. The second question of the test was about the *west_of* relation in the same setting. The comparison is shown in Figure 21. Even in this case, the two results are much closer to each other: we can notice a tendency of users at picking up more "precise" relations, discarding buildings that are in intermediate directions. The third question concerned the deictic frame of reference; the comparison of results with querying the application is illustrated in Figure 22. It is interesting to note that in this situation users were more permissive to include some buildings to the left, even if at lower percentages. The fourth question still deals with the deictic frame of reference in the case of the relation *right_of*. The results are compared in Figure 23. In this case, we obtained a good match between the application and user tests, since all the results of the application were chosen at least by one third of the users.

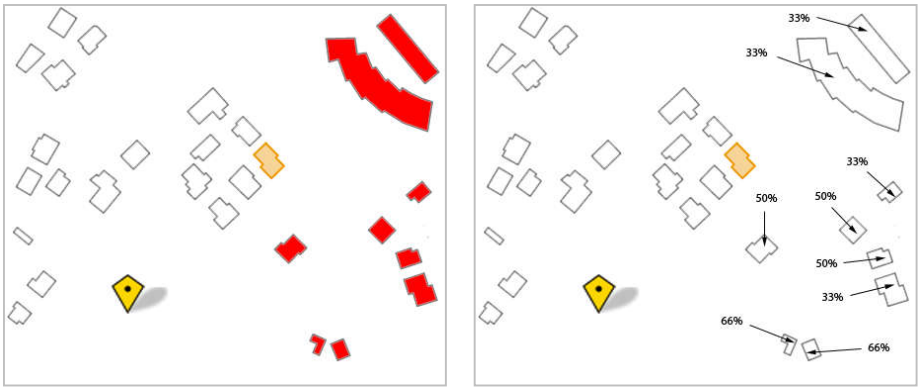



(**a**) (**b**)

**Figure 21**. Results of the *west_of* relation with respect to the selected building: (**a**) from the application; (**b**) from the user tests.


(**a**) (**b**)

**Figure 22**. Results of the *front* relation with respect to the selected building: (**a**) from the application; (**b**) from the user tests.

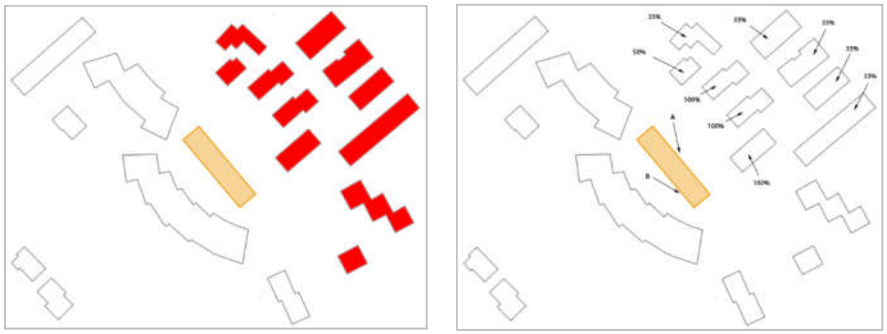



(**a**) (**b**)

**Figure 23**. Results of the *right_of* relation with respect to the selected building: (**a**) from the application; (**b**) from the user tests.

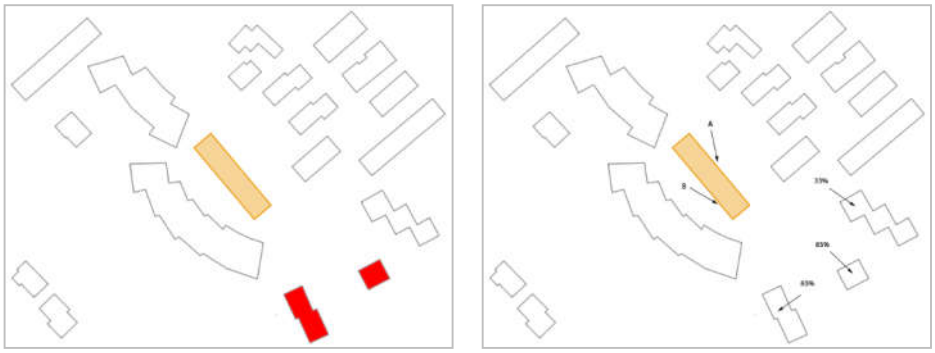
The last two questions proposed to the participants concern the geometric frame of reference. Figure 24 shows the results for the *front* relation. The buildings included by the application are broader than the user tests: it is the long shape of the reference building that influences, in this case,

the area that is considered to be in front by our application framework. It is interesting to note that the percentages of user answers are higher for closer buildings than for further away buildings. Therefore, closer buildings have a stronger influence on the judgement of directions. The last question concerns the relation *rightside* of the geometric frame of reference. Figure 25 shows the results.



(**a**) (**b**)

**Figure 24**. Results of the *front* relation with respect to the selected building: (**a**) from the application; (**b**) from the user tests.



(**a**) (**b**)

**Figure 25**. Results of the *rightside* relation with respect to the selected building: (**a**) from the application; (**b**) from the user tests.

## 6. Conclusions

Directional relations can be applied in every context related to spatial data, from spatial databases to navigation systems [26,27] or crowdsourced data [28,29]. Despite the importance of directional relations, geospatial standards do not include them among the set of available spatial relations. This lack can be explained perhaps because the theory behind directional relations is quite fragmented and little consensus appears on their definition and models. More complexity is added by the fact that directional relations need to be put in a context of a frame of reference to be disambiguated and quite a number of different frames of reference can be envisaged. Projective relations [24] can be considered to be a computational basis for directional relations: in [21], a mapping between projective relations and several useful frames of reference is proposed. Correspondingly, sets of directional relations are proposed for each frame of reference.

In this paper, we describe an application framework in Java that implements the directional model in a three—layered architecture. We develop a case study in which several kinds of queries involving directional relations are posed against the system. In the last part of the paper, we use the case study to outline a user test of the cognitive adequacy of frames of reference and directional relations of the model. In fact, the model proposed in [21] has never been tested with users and a

deeper evaluation is the goal of further research. By using the application framework, it was possible to compare the output of the application framework with answers given users. We found a good correspondence of the model with human cognized directional relations, and, where differences were found, the results can be used for improving the model in future work.

**Author Contributions:** Conceptualization, E.C.; Data curation, G.B.; Formal analysis, E.C.; Methodology, E.C.; Resources, G.B.; Software, G.B.; Supervision, E.C.; Validation, E.C.; Writing—original draft, E.C.; Writing—review & editing, E.C.

**Funding:** Authors acknowledge the Department of Industrial and Information Engineering and Economics, University of L'Aquila, for covering the costs of this publication.

**Conflicts of Interest:** The authors declare no conflicts of interest

## References

1. Egenhofer, M.J.; Herring, J.R. *Categorizing Binary Topological Relationships between Regions, Lines, and Points in Geographic Databases*; Technical Report; Department of Surveying Engineering, University of Maine: Orono, ME, USA, 1990; p. 28.
2. Clementini, E.; Cohn, A.G. RCC*-9 and CBM*. In *Geographic Information Science, Proceedings of the 8th International Conference, GIScience 2014, Vienna, Austria, 24–26 September 2014*; Duckham, M., Pebesma, E., Stewart, K., Frank, A.U., Eds.; Springer: Berlin, Germany, 2014; Volume LNCS 8728, pp. 349–365.
3. Tarquini, F.; Clementini, E. Spatial relations between classes as integrity constraints. *Trans. GIS* **2008**, *12*, 45–57, doi:10.1111/j.1467-9671.2008.01134.x.
4. Clementini, E. Ontological Impedance in 3D Semantic Data Modeling. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XXXVIII-4, Part W15, Proceedings of the ISPRS: 5th 3D GeoInfo Conference, Berlin, Germany, 3–4 November 2010*; Kolbe, T.H., König, G., Nagel, C., Eds.; Shaker Verlag GmbH: Aachen, Germany, 2010; pp. 97–100.
5. Bucher, B.; Falquet, G.; Clementini, E.; Sester, M. Towards a typology of spatial relations and properties for urban applications. In *3u3d2012: Usage, Usability, and Utility of 3D City Models, Proceedings of the European Cost Action TU801 Final Conference, Nantes, France, 29–31 October 2012*; EDP Sciences: Les Ulis, France, 2012.
6. Billen, R.; Clementini, E. Introducing a reasoning system based on ternary projective relations. In *Developments in Spatial Data Handling, 11th International Symposium on Spatial Data Handling*; Fisher, P., Ed.; Springer: Berlin, Germany, 2005; pp. 381–394.
7. Billen, R.; Clementini, E. Semantics of collinearity among regions. In *On the Move to Meaningful Internet Systems 2005: OTM Workshops, Proceedings of the 1st International Workshop on Semantic-based Geographical Information Systems (SeBGIS'05), Agia Napa, Cyprus, 31 October–4 November 2005*; Meersman, R., Tari, Z., Herrero, P., Eds.; Springer: Berlin, Germany, 2005; Volume 3762, pp. 1066–1076.
8. Billen, R.; Clementini, E. Projective relations in a 3D environment. In *Geographic Information Science, Proceedings of the 4th International Conference, GIScience 2006, Münster, Germany, 20–23 September 2006*; Raubal, M., Miller, H., Frank, A., Goodchild, M., Eds.; Springer: Berlin, Germany, 2006; Volume 4197, pp. 18–32.
9. Clementini, E. Projective Relations on the Sphere. In *Advances in Conceptual Modeling—Challenges and Opportunities—ER 2008 Workshops, Proceedings of the 2nd International Workshop on Semantic and Conceptual Issues in GIS (SeCoGIS 2008), Barcelona, Spain, 20–23 October*; Song, I.-Y., Ed.; Springer: Berlin/Heidelberg, Germany, 2008; Volume 5232, pp. 313–322.
10. Bartie, P.; Clementini, E.; Reitsma, F. A Qualitative Model for Describing the Arrangement of Visible Cityscape Objects from an Egocentric Viewpoint. *Comput. Environ. Urban Syst.* **2013**, *38*, 21–34.
11. Fogliaroni, P.; Clementini, E. Modeling Visibility in 3D Space: A Qualitative Frame of Reference. In *3D Geoinformation Science: The Selected Papers of the 3D GeoInfo 2014*; Breunig, M., Al-Doori, M., Butwilowski, E., Kuper, P.V., Benner, J., Haefele, K.H., Eds.; Springer: Berlin, Germany, 2015; Volume LNG&C, pp. 243–258.
12. Bartie, P.; Reitsma, F.; Clementini, E.; Kingham, S. Referring Expressions in Location Based Services: The case of the 'Opposite' Relation. In *Advances in Conceptual Modeling. Recent Developments and New Directions—ER 2011 Workshops, Proceedings of the Fifth International Workshop on Semantic and Conceptual Issues in GIS (SeCoGIS 2011), Brussels, Belgium, 1 November* 2011; De Troyer, O., Bauzer Medeiros, C., Billen, R., Hallot,

P., Simitsis, A., Van Mingroot, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; Volume LNCS 6999, pp. 231–240.
13. Liu, X.; Shekhar, S.; Chawla, S. Object-based directional query processing in spatial databases. *IEEE Trans. Knowl. Data Eng.* **2003**, *15*, 295–304.
14. Worboys, M.; Duckham, M.; Kulik, L. Commonsense notions of proximity and direction in environmental space. *Spat. Cogn. Comput.* **2004**, *4*, 285–312.
15. Frank, A. Qualitative Spatial Reasoning about Cardinal Directions. In Proceedings of the Tenth International Symposium on Computer-Assisted Cartography (AUTO CARTO 10), Baltimore, MD, USA, 25–28 March 1991; pp. 148–167.
16. Goyal, R.; Egenhofer, M.J. The Direction-Relation Matrix: A Representation of Direction Relations for Extended Spatial Objects. In Proceedings of the UCGIS Annual Assembly and Summer Retreat, Bar Harbor, ME, USA, 15–21 June 1997.
17. Kulik, L.; Klippel, A. Reasoning about Cardinal Directions Using Grids as Qualitative Geographic Coordinates. In *Spatial Information Theory: Cognitive and Computational Foundations of Geographic Information Science, Proceedings of the International Conference COSIT'99, 25–29 August 1999, Stade, Germany*; Freksa, C., Mark, D.M., Eds.; Springer: Berlin, Germany, 1999; Volume 1661, pp. 205–220.
18. Levinson, S.C. Frames of reference and Molyneux's question: Crosslinguistic evidence. In *Language and Space*; Bloom, P., Peterson, M.A., Nadel, L., Garrett, M.F., Eds.; The MIT Press: Cambridge, MA, USA, 1996; pp. 109–169.
19. Retz-Schmidt, G. Various Views on Spatial Prepositions. *AI Mag.* **1988**, *9*, 95–105.
20. Frank, A.U. Formal Models for Cognition—Taxonomy of Spatial Location Description and Frames of Reference. In *Spatial Cognition: An Interdisciplinary Approach to Representing and Processing Spatial Knowledge*; Freksa, C., Habel, C., Wender, K.F., Eds.; Springer: Berlin, Germany, 1998; Volume 1404, pp. 293–312.
21. Clementini, E. Directional relations and frames of reference. *GeoInformatica* **2013**, *17*, 235–255, doi:10.1007/s10707-011-0147-2.
22. Hua, H.; Renz, J.; Ge, X. Qualitative Representation and Reasoning over Direction Relations across Different Frames of Reference. In *Principles of Knowledge Representation and Reasoning, Proceedings of the Sixteenth International Conference, KR 2018, Tempe, AZ, USA, 30 October–2 Vovember 2018*; AAAI Press: Palo Alto, CA, USA, 2018; pp. 551–560.
23. Scheider, S.; Hahn, J.; Weiser, P.; Kuhn, W. Computing with cognitive spatial frames of reference in GIS. *Trans. GIS* **2018**, *22*, 1083–1104.
24. Clementini, E.; Billen, R. Modeling and computing ternary projective relations between regions. *IEEE Trans. Knowl. Data Eng.* **2006**, *18*, 799–814.
25. Clementini, E.; Billen, R. Projective Relations. In *Encyclopedia of GIS*, 2nd ed.; Shekhar, S., Xiong, H., Zhou, X., Eds.; Springer International Publishing: New York, NY, USA, 2016; pp. 1–10.
26. Mortari, F.; Zlatanova, S.; Liu, L.; Clementini, E. "Improved Geometric Network Model" (IGNM): A novel approach for deriving Connectivity Graphs for Indoor Navigation. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.* **2014**, *II-4*, 45–51.
27. Russo, D.; Zlatanova, S.; Clementini, E. Route Directions Generation using Visible Landmarks. In Proceedings of the Sixth ACM SIGSpatial International Workshop on Indoor Spatial Awareness (ISA 2014), Tampa, FL, USA, 8–12 September 2014; Claramunt, C., Li, K.-J., Zlatanova, S., Eds.; ACM: New York, NY, USA, 2014; pp. 1–8.
28. Fogliaroni, P.; D'Antonio, F.; Clementini, E. Data trustworthiness and user reputation as indicators of VGI quality. *Geo-Spat. Inf. Sci.* **2018**, *21*, 213–233.
29. Martella, R.; Kray, C.; Clementini, E. A Gamification Framework for Volunteered Geographic Information. In *AGILE 2015—Geographic Information Science as an Enabler of Smarter Cities and Communities*; Bação, F., Santos, M.Y., Painho, M., Eds.; Springer: Berlin, Germany, 2015; Volume Lecture Notes in Geoinformation and Cartography, pp. 73–89.