An Architecture to Manipulate Continuous Fields in GIS Applications

Luis Polasek, Arturo Zambrano, Silvia Gordillo

LIFIA- Facultad de Informática, UNLP

CC 11 (1900) La Plata, Buenos Aires, Argentina

[pola, arturo, gordillo]@sol.info.unlp.edu.ar

Abstract

It is now usual that GIS applications need to define and handle continuous information. This is of extreme relevance in areas like meteorology and environmental applications. GIS do not have tools that allow the use of this type of information, which often prevents to make a complete analysis. In this work we describe an object-oriented architecture that allows us to design and manipulate continuous data.

1. Introduction

Geographic Information Systems (GIS) provide the necessary tools to manipulate and to analyze spatial entities, that is to say, which have a position in the space. In general, GIS handle two different data models, the vector and the raster model [5, 1]. Whereas the vector model recognizes individual objects that have a topology (point, line or polygon) and a position, the raster model defines a region, and then this region is divided into smaller and indivisible sub-regions; each sub-region having a value that describes a relevant attribute in that area.

Although some operations are more complex than others, any of both models can be used in a GIS in an efficient way; even there are systems in which it is possible to combine information of both models.

A great variety of applications exists where the information that is required is in essence "continuous"; for example any weather related application must have the capacity to analyze continuous phenomena, like the temperature, the humidity, etc. Although it is certain that mathematical models exist to deal with the continuity of phenomena, it is also certain that good computational models do not exist to deal with this type of information [4].

A typical example of continuous data can be a wave or any analogic signal that varies on time. Mathematically, $f(t)$ is a continuous function, where t is the time. If we took two moments t_0 and t_1 ($t_0 \neq t_1$), we have:

$$f(t_0) = a \text{ and } f(t_1) = b \text{ and we suppose that } a < b$$

Since f is a continuous function we know f takes all the possible values in [a, b] when f is evaluated in the interval $[t_0, t_1]$.

The same idea can be extended from one to two dimensions, defining the domain of the function in a coordinate plane. From this point of view we can define continuous phenomena in the context of geographic applications, these phenomena are known as continuous fields.

If we think that f represents the value corresponding to some physical phenomenon that can be measured in a terrestrial region, it is possible to define a function that returns the value of the phenomenon associated to any point p_i that belongs to this region.

The data represented in the continuous field can be any of these two types:

i Scalar: returned values by the function are numbers.

ii Vectorial: returned values by the function are vectors.

A scalar value is an uni-dimensional datum, for example a number.

A vectorial value is a n-dimensional datum, for example the gradient in a point, is denoted by a tuple like this:

$$(d_1, d_2,..., d_n)$$

The usual way to manipulate this kind of information is to make a sampling on the data. Values associated to the phenomenon are taken in some positions within the study area and the rest of the needed values are calculated through some estimation method; this estimation method uses the values of the sample as starting points for the calculation.

In most of current GIS applications, the process previously depicted is performed outside the system. Standard GIS are capable to represent the values of the sample, but unfortunately, they have not a mechanism to make estimations and to operate with this kind of information. In those systems the acquired points are kept in a database. When it is necessary to make a query referring to a value of the function in a position that does not belong to the sample, the needed values are exported to some specific calculus application to calculate the value by means of interpolation, then the produced values are imported. Eventually, they can be incorporated to the database (to accelerate future queries).

An architecture to specify continuous fields in GIS applications has been presented by *Gordillo* [3]. This architecture takes into account all the aspects we mentioned before. The objective of the present work is to define a series of essential operations on the continuous fields and in such sense, to extend this architecture in a way to obtain the skill of manipulating the continuous information.

This paper is organized as follows: In section 2 we describe the architecture presented in *Gordillo* [3] that constitutes the starting point to define the operations on continuous fields. In section 3 we characterize the different kinds of operations on continuous fields. In section 4 we introduce an architecture to solve requests and their operations on continuous fields. Next, in section 5 we show how the n-ary operations are modeled. In section 6 we integrate all previous results that constitutes the request architecture. Finally, in section 7 we present the conclusions and the future work.

2. Representing Continuous Fields

The model presented by *Gordillo* [3] is an object-oriented model that defines a set of classes and relations to specify continuous fields in the context of geographic applications. Figure 1 shows this architecture.

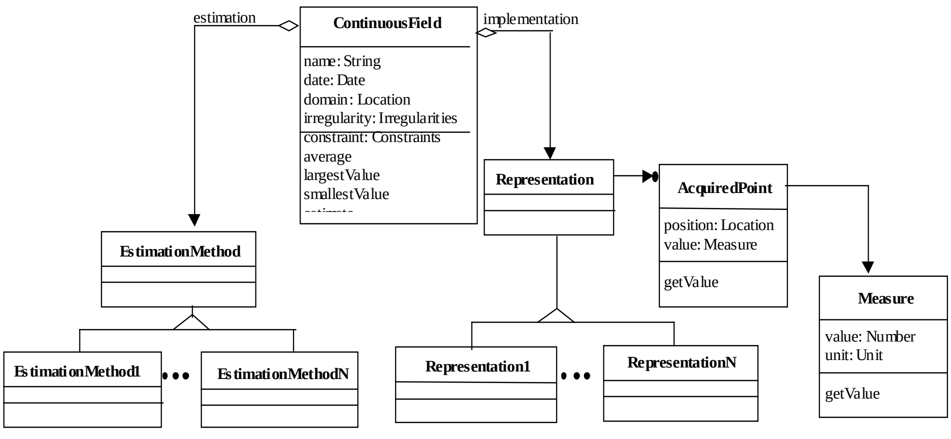

Figure 1. Object Oriented Architecture for fields.

The *ContinuousField* class contains information like the phenomenon that is represented, the date in which it was created and the domain that represents the region under study. In the architecture it is possible to represent continuous field in several ways; for example by a regular grid where the points of the sample are distributed in the grid vertices. Another possible representation could be an irregular grid, or a TIN (Triangulated Irregular Network). In order to model the different representations that a continuous field can adopt in this architecture it is used the pattern Bridge [2]. A continuous field can also vary the form in which the values of the non-sampled points are calculated. The way in which those values are calculated is based on an estimation method. In this case the Strategy design pattern of [2] was applied.

3. Continuous Fields Requests and Operations

One way to manipulate continuous fields is through **requests** based on the application of a series of basic operations. This set of operations should be rich enough, so that, when they are combined it is possible to make any kind of manipulation involving continuous data. In this section we discuss how to define these operations involved in a request.

Many of geographical phenomena must be modeled as continuous fields. For example, features such as temperature, humidity, pressure and height are continuous data and therefore it is needed to define them as continuous fields and associate to them an estimation method, a representation, constraints, irregularities and so on. Many often, each one of these features are defined in different geographical regions (overlapped or not), sampled points do not have the same position, and even the chosen representations could be different. As a consequence of these characteristics, it is necessary to be able to operate with continuous fields that are not defined in the same spatial model [4] and which are not describing the same domain. Moreover, many times estimation methods are also dependent of the representation chosen for each field.

In order to perform operations considering all previous aspects, some spatial and mathematical problems have to be solved.

Normally, a request is a complex sentence to retrieve some information from continuous fields. Requests can be decomposed into single operations, whose results are combined to solve the request. In each step of this decomposition, an operation is applied to one or more continuous fields (and possibly to other parameters); this operation, produces a new continuous field that can be taken as the input in the next step.

That leads us to think that a request is composed of simpler ones, which, in turn, can be composed of other simpler ones, and so on. This relation could have as many levels as necessary to obtain the information based on the basic operations defined for continuous fields.

Making a request on continuous fields implies to operate with them. These operations vary according to the complexity, the number of continuous fields involved and the kind of data that will be operated.

Is necessary then, to establish all possible operations that can be carried out on continuous fields, and the way they are solved.

The operations can take as input not only continuous fields but also other kind of data, and we can divide them in two groups:

- Unary Operations: involves one continuous field; some examples of unary operations are:

Evaluate:	returns the value associated to a particular location (it can be an existing value or an estimated one).
Average:	returns the average value of the field.
Maximum:	returns the maximum value of the field.
Minimum:	returns the minimum value of the field.
Slope:	returns the slope associated to a particular location.
Select:	returns a new continuous field whose domain matches with a given condition.
Convex Hull:	returns the convex hull of the sample.
Representation Change:	returns a new continuous field with the same sample and domain, with a different representation. The representation change can imply the change of the estimation method.

- N-ary Operations: two or more continuous fields are involved; some examples of n-nary operations are:

Average:	returns a continuous field where each value associated to a position is the average among the values associated to the same position in each continuous field.
Maximum:	returns a continuous field where each value associated to a position is the maximum among the values associated to the same position in each continuous field.
Minimum:	returns a continuous field where each value associated to a position is the minimum among the values associated to the same position in each continuous field.
Arithmetic Operation:	these operations are those defined by *Tomlin* [6].
Union:	returns a continuous field which sample is the union of the samples of several continuous fields.

4. An Architecture to Solve Requests and their Operations on Continuous Fields

The architecture defined in this section allows us to retrieve information from continuous fields, based on the ideas discussed in the previous section. The goal is to show how to manage different complexity levels by using composition and delegation.

4.1. Simple and Complex Information Requests

Retrieving information from continuous fields may result in applying one or several operations on a continuous field set; and this situation can turn into a very complex computation process.

In order to solve the complexity involved in the resolution, we propose to decompose the information retrieval in a set of simpler operations that are combined in order to obtain the final result, by solving each step and combining the results. In this way, partial results are obtained and then they are combined with other results to achieve the final result.

In the architecture we define a class (the *Request* class) with all the necessary knowledge to retrieve the desired information. Given the case that a request needs the previous evaluation of another ones, this would be part of its knowledge, as well as the way that they will be returned. In other cases this decomposition will not be necessary, neither the associate knowledge; these cases are those representing simple operations.

It would be desirable that these two kinds of information requests were dealt in a uniform way in the architecture.

We see that this consideration can be reached by using the structure defined by the pattern Composite [2]. The general structure is shown in Figure 2.

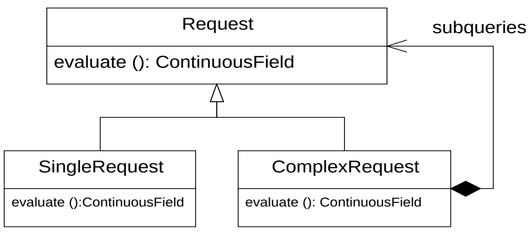

Figure 2. Hierarchy for Requests

Where:

1. A *SingleRequest* retrieves a continuous field associated with the operation to be solved. It wraps a continuous field and returns it when asked to evaluate itself.
2. A *ComplexRequest* take as the input the result of the evaluation of other *Requests* and then it applies some operation. For accomplishing this, it needs to know which requests will produce the input fields that are necessaries to the operation. It takes several continuous

fields, which are the result of other operations, applies the operation it represents, and returns the result of the operation (that will be a continuous field).

4.2. Operations

In this section we analyze the alternatives to model operations on continuous fields.

We have defined a classification that divide the modeling of operations on continuous fields in two types:

- Simple operations, that involve one continuous field and possibly some simple parameters. These operations can be solved by the continuous field itself.
- Complex operations, that involve at least two continuous fields. To solve these operations it is necessary to take into account all different alternatives that can rise from the combination among the components of different continuous fields, such as the representation, the estimation method, the domain and the sample of the input continuous field. This kind of operations requires the interaction among several objects having different responsibilities in order to solve all aspects involved during the resolution process.

The former corresponds to unary operations and the later to n-ary operations, that we have defined in an earlier section.

Since unary operations do not require a complex processing, the *ContinuousField* class can easily solve them. So we are not going to extend in this topic and we will describe in detail the n-ary operations modeling which are not as simple as the unary operation model.

4.3. N-ary Operations

The reason to decouple the resolution of these operations from the continuous field is to avoid that the continuous field class becomes in a very complex and hardly to extend class. Several aspects must be solved and the combination of different alternatives in the use of representations, estimation methods and domains generate a great number of responsibilities to manage. Instead, it is possible to encapsulate the resolution of the operation in a class specifically defined to manage all the knowledge and interactions needed in each case.

In order to provide the coordination among all participating objects, we have defined an object called *RequestEvaluator*, which is responsible for coordinating steps to obtain the operation's result into a *Request*, we will refer to it later.

In the next section, we will analyze and describe several aspects involved in n-ary operation work out.

5. Solving N-ary Operations

As we previously said, we must solve several aspects in order to make an n-ary operation on continuous fields. In this section we will explain how the domain, sample points position and sample points values are calculated in order to solve a n-ary operation.

5.1. Obtaining the Resulting Continuous Field's Domain

The objective is to define the geographical region in which the resulting continuous field will be defined (the new domain). In order to determine this region we use three operations: **union, intersection** and **difference** applied to the polygons defined by the continuous field's domain constituting the input.

Domain Union The resulting domain is composed by the union of all polygons of all fields in the input. This process can generate a domain that covers areas not covered in each particular field, so estimating in those places not supported by the input fields could increase errors in the estimated values.

Domain Intersection Only the area covered by all input fields will be the resulting field domain. Since all fields support the domain, there would be no increase in the estimation errors when performing operations.

Domain Difference The domain in the resulting continuous fields will be the one that is the difference between all input field's domains. This way of computing the field's domain has the same characteristics as *Domain Union*.

The next figure (Figure 3) displays three different domains built from the two input continuos fields domains.

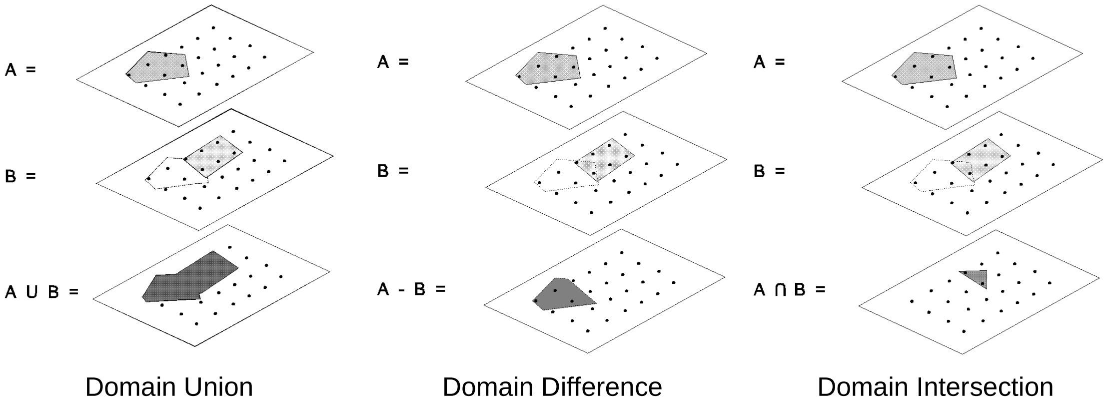

Figure 3. Zone Operation

To establish the domain of the resulting field we have used the design pattern Strategy [2]. This pattern encapsulates several strategies that allow us to define how the resulting domain will be obtained.

In Figure 4, we show the relation between a *RequestOperationEvaluator* and the *ZoneOperation* hierarchy. In this way we can configure the evaluator with the proper *ZoneOperation* in order to obtain the resulting continuous field's domain.

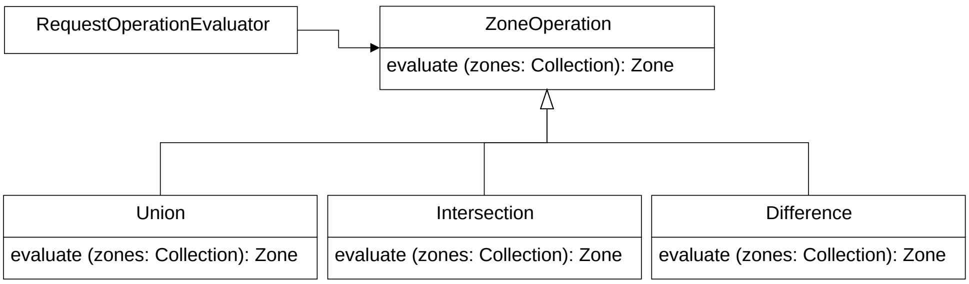

Figure 4. Hierarchy for ZoneOperations

5.2. Obtaining the Sample Points for the Resulting Field

Different samples corresponding to different continuous fields can have points acquired in non-coincident positions. It is important to define a strategy in order to generate (or select) the points which will be part of the sample in the resulting continuous field. Obviously, a value on an acquired sample point has better precision than an estimated value. The selected strategy affects directly the number of estimations required for a given operation.

The density and the precision of the new sample depend on the selected strategy. Here we enounce some features of each strategy (we use the term precision to denote that a value calculated from values belonging to sampled points is more precise than another calculated from estimated values).

Sample Union The resulting sample is composed by all points of all fields in the input. In this case, when a value corresponding to a specific location does not exist in the other fields, it is estimated according to the estimation method defined for the field. This process generates a dense sample but many points have to be estimated to obtain the result, this can produce a sample with poor precision. This density condition could benefit to the estimation method, since all estimation methods work better when there are many points.

Sample Intersection Only those points acquired in all input fields will be taken into account to perform the operation. The density of the resulting sample is lower than the previous case but there are not estimated values in the result, it is not necessary any data interpolation since all selected points are sampled in all input continuous fields. Since there are not interpolations to obtain the new sample, it will have a high quality but lower density.

The density of the resulting sample is lower or equal than the samples of the input fields.

Sample Difference The points in the resulting continuous fields would be those that are in one input field but not in the other input fields.

Arbitrarian Sample The points of the result are chosen arbitrarily. In this case the need to estimate certain points, the density and the precision of the sample depends on the selected points

To establish which points will be part of the resulting field sample we have also used the design pattern Strategy [2]. This pattern allows us to encapsulate the strategies that define the new sample points.

The client object (*RequestOperationEvaluator*) will use these strategies when it needs to determine which points will belong to the resulting field sample.

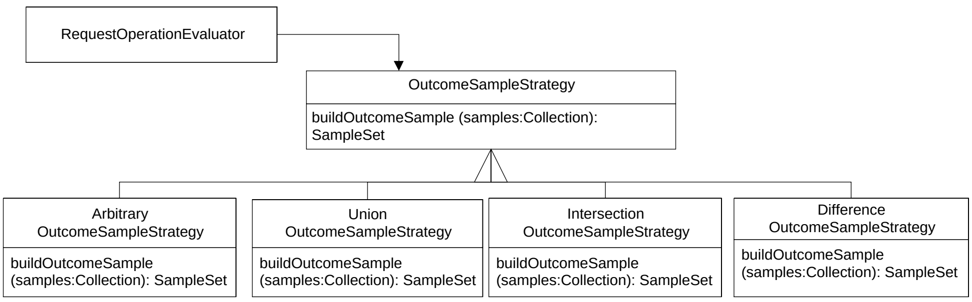

Figure 5. Hierarchy for OutcomeSampleStrategy

In previous figure, *RequestEvaluator*, is the responsible for performing the operation among several continuous fields, and *OutcomeSampleStrategy*, is responsible for establishing the position of the points that belong to the resulting continuous field sample.

In this way, it is easy to configure the object that performs the operation (a request) in order to select the desired positions. The evaluator requests to its instance of *OutcomeSampleStrategy* to select those points that will be part of the resulting continuous field. Furthermore, this solution simplifies the addition of new strategies to select points.

- *ArbitraryOutcomeSampleStrategy:* returns a set of arbitrary selected positions, independently of input continuous field samples form.
- *UnionOutcomeSampleStrategy:* returns the resulting positions set from the union among all input continuous fields samples.
- *IntersectionOutcomeSampleStrategy:* returns the resulting positions set from the intersection among all input continuous fields samples.
- *DifferenceOutcomeSampleStrategy:* returns the resulting positions set from the difference among all input continuous fields samples.

5.3. Obtaining Values that Conform the Resulting Field Sample

When you have chosen the points that will be in the resulting sample it is necessary to determine the associated value for each of them. In order to accomplish this task, the resulting values for the new sample are obtained from the application of a function to values taken from all the input continuous fields that constitute the input. Each value belonging to the resulting sample has the form:

$$y = f(z_1, \ldots, z_n)$$

Where z_i is a value associated to the continuous field CC_i on the position *p*, and *y* is the value that will be associated to the position *p* in the resulting continuous field.

Graphically

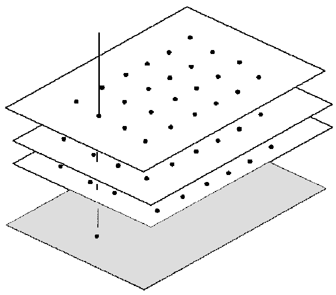

Figure 6. Punctual Operations take values from the same position.

So, we can see that each $z_i = G_i(p)$, where *p* is the position we are working on and G_i is the phenomenon value associated to the position *p* in the continuous field *i*.

The structure shown in Figure 6 allows the evaluator to interact with any punctual operation, since all operations have the same protocol (they are a polymorphic hierarchy). In this way the model increases its flexibility and extensibility in order to support the incorporation of new operations.

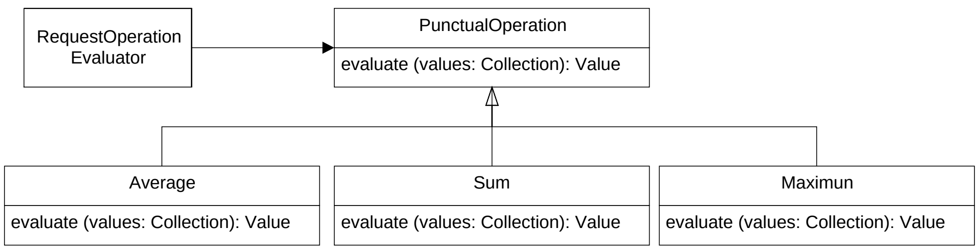

Figure 7. Hierarchy for Punctual Operations

5.4. The Evaluator

So far, we have shown how we solve the different aspects referring to continuous field operation in a isolated way, taking into account different alternatives that appears in each case. Which remains to do is to define the way that all this solutions are integrated.

The idea is to have an object that is responsible for coordinating how the problem will be solved, by:

- Establishing the resulting continuous field's domain.
- Defining the resulting continuous field's sample.
- Selecting a representation type.
- Establishing the estimation method according to the previous components.

The *RequestEvaluator* is a configurable object that permits to define how these aspects will be solved in each step of a request.

Besides, there exists another evaluator subclass (*InternalOperationEvaluator*) which permits the Request to make unary operations. This is why the Request always interacts with a *RequestEvaluator* and, as we said before, the continuous fields solves the unary operations. The only thing that the *InternalOperationEvaluator* does is to forward the requests to the corresponding continuous field; achieving the execution of this operation by the Request in a transparent way.

The evaluator permits to make an *operation* on one or more continuous fields in a request.

6. Request Architecture

At this point some doubts can rise about the responsibilities of the objects that participate in the architecture. These objects are: the request (represented by the *Request* class), the evaluators (represented by the *RequestEvaluator* class) and the operations (represented by the *Operation* class).

Recall that the requests were modeled using the Composite pattern and we have *SimpleRequest* and *ComplexRequest* (see Figure 2).

The ComplexRequest collaborates with its sub-requests to obtain its input continuous fields. When it has these fields, it asks to its associated evaluator to evaluate himself.

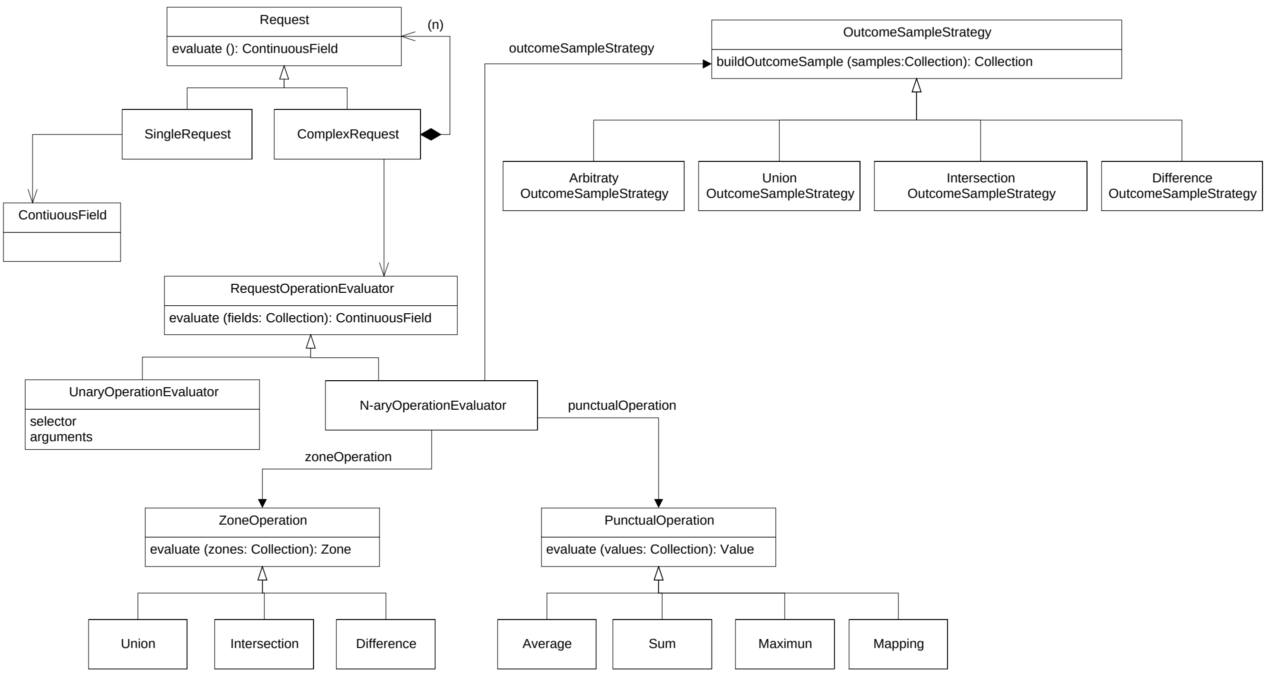

Figure 8. Request Class Architecture

From all what we have explained previously we can see that:

- the request can be simple or complex.
- the complex request references to an evaluator that permits to make operations between continuous fields.
- the evaluators are sub classified according to how the operations between one or more continuous fields are made.
- the evaluators are aided by an *OutcomeSampleStrategy* object to decide how the resulting continuos field sample will be.
- the evaluators are aided by a *PunctualOperation* object to calculate the values associated to each sample point.
- the evaluators are aided by a *ZoneOperation* object to determine the resulting domain.

In the particular case of the *N-aryOperation*Evaluator, it takes the input continuous fields that are supplied by the *ComplexRequest* and produces a new continuous field collaborating with an instance of *ZoneOperation* (to establish the resulting continuous field domain), an instance of the *OutcomeSampleStrategy* (to define the resulting sample points position) and with an instance of *PunctualOperation* (to calculate the associated value of each point in the resulting continuous field sample).

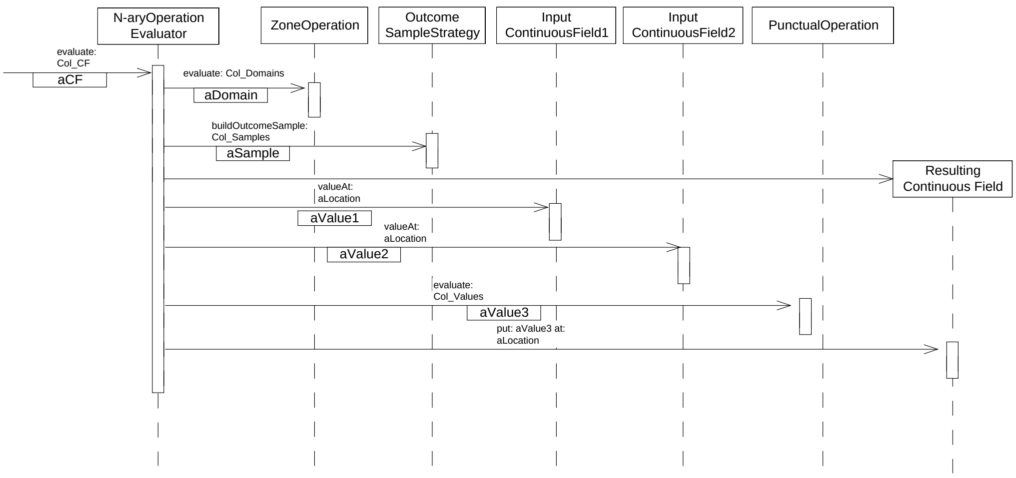

Figure 9. Interaction Diagram

```
N-aryOperationEvaluator>>evaluate: aCollection
    " aCollection is a set of Continuous Fileds "
    | resultingField representations inputValues domains |
        resultingField := ContinuousField new.
        Domains:= aCollection collect: [:aContinuousField | aContiuousField domain ].
        Representations:= aCollection collect: [:aContinuousField |
                                                  AContiuousField representation].
        ResultingField domain: (self zoneOperation evaluate: domains).
        ResultingField representation: (PointGrid new).
        self outcomeSampleStrategy
                        buildOutcomeSample: representations
                        for: resultingField representation.
        ResultingField representation sample
                do:
                    [:aValuePair |
                    inputValues := aCollection collect: [:aCF |
                                                  aCF evaluate: aValuePair location].
                    aValuePair value: (self operation evaluate: inputValues)].
        ResultingField evaluator: self outcomeEstimationMethod.
        ^resultingField
```

Sample Code

The code in bold shows how a value in the resulting continuos field is calculated, this operation is done with all points in the sample.

It is important to note that as a consequence of using a punctual operation, a field with a new sample is obtained, it is generated on-line and no references to the original samples is saved.

7. Conclusions and Future Work

In this work we have exposed an architecture to solve the problem of accomplishing complex requests from continuous fields represented as point grids or TINs. Previously, we mentioned that exist different types of representations for continuous fields. Now, we are working on the extension of the presented architecture to support operations that permits requests to continuous fields with different representations, like Tesellations and Level Curves. The major problem that presents is the difference in the spatial models of the continuous fields, which can be solved using some of the rules presented by *Kemp* [4].

Finally an object-oriented implementation will be provided, as part of a framework for geographic applications.

8. References

1. Aronoff, S. “Geographic Information Systems : A Management Perspective”. WDL Publications, 1991

2. Gamma, E. ; Helm, R.; Johnson, R.; Vlissides, J. "Design Patterns. Elements of reusable Object-Oriented Software". Addison Wesley, 1995.

3. Gordillo, S.; Balaguer, F. “Refining an object-oriented GIS design model: Topologies and Field Data”. ACM-GIS’98. November 6-7, 1998. Washington, D.C., USA.

4. Kemp, K. “Fields as a Framework for integrating GIS and environmental process models”.
   Part 1: “Representing spatial continuity”.
   Part 2: “Specifying field variables”.
   Transactions in GIS, 1997, vol. 1, nº 3, p. 219-234 and 235-246.

5. Laurini, R.; Thompson, D. “Fundamentals of Spatial Information Systems”. Academic Press Professional, 1993.

6. Tomlin, C. D. “Geographic Information Systems and Cartographic Modeling”. Prentice-Hall, Inc., 1990. New Jersey.