Final report for RGI-116 WP4.1
Towards standardised dynamic and
3D field representations

Representing Continuous Geographical Phenomena with `FieldGML`

Hugo Ledoux
`h.ledoux@tudelft.nl`
TU Delft

Preliminary version—December 15, 2008

Preliminary version—December 15, 2008

While we can affirm that the representation, storage and exchange of two-dimensional objects (vector data) in GIS is solved (at least if we consider the *de facto* standards *shapefile* and GML), the same cannot be said for fields. Among the GIS community, most people assume that fields are synonymous with raster structures, and thus only representations for these are being used in practice (many formats exist) and have been standardised.

In this report, I present a new GML-based representation for fields in 2D and 3D, one that permits us to represent not only rasters, but also fields in any other forms. This is achieved by storing the original samples of the field, alongside the interpolation method used to reconstruct the field. The solution, called *FieldGML*, is based on current standards, is flexible, extensible and is also more appropriate than raster structures to model the kind of datasets found in GIS-related applications.

For more information about FieldGML:

`http://www.gdmc.nl/ledoux/fieldgml.html`

Note: This document is the final report for the WP4.1.1 "Towards standardised dynamic and 3D field representations" of the *Ruimte voor Geo-Informatie* RGI-116 project, entitled "Internationale geostandaarden". The projects aimed at making recommendations for the standardisation of dynamic and 3D fields. Many standards exist for static objects and fields (e.g. ISO 19107:2003 Spatial schema; ISO 19141 Schema for moving features; ISO 19123 Schema for coverage geometry and functions) but there seems to be a gap in the standards for fields in more than two dimensions and/or when the temporal factor is taken into consideration. The project is driven by the increasing use and collection of such datasets in many disciplines related to the Earth, most notably among the oceanographic and atmospheric communities. This project was two-fold, in the sense that it had to cover:

1. the abstract specifications of fields (in 2D and 3D). That means clearly defining what fields are, how they can be represented in a computer, and also how the temporal dimension influences the representations.

2. the implementation specifications, i.e. what formats and structures should be used to store fields.

Contents

1 Introduction

There exist two documents related to the standardisation of fields: 'Schema for coverage geometry and functions' (ISO 19123) (ISO, 2005b), and the OGC document with the same title (OGC, 2007b). As highlighted in this report, these contain several shortcomings, weaknesses and omissions for the representation of fields, and should be updated. These shortcomings probably come from the fact that the definition of a field itself changes from discipline to discipline, and that the issues can be seen from a philosophic, conceptual or implementation point of view (Peuquet et al., 1999). There is also much confusion among users between spatial models, data structures, and spatial concepts (Frank, 1992). While in the GIS jargon object- and field-views of space are often synonymous with respectively vector and raster models, Goodchild (1992), among others, explains that this is simply false as both views can be stored with either model. Put on top of that that fields are by definition something continuous—and that computers are discrete machines—and one can start understanding the confusion among users. These difficulties, coupled with the shortcomings of current standards, result in a situation where the standards are barely used in practice, except for raster format.

This report presents a new alternative for representing fields called *FieldGML*. This is a generic solution based on current standards (i.e. GML), and permits us to efficiently store and exchange field-based geographic information, and not only rasters. The main idea behind this representation is that instead of storing explicitly grids or tessellations, we store the data that were collected to study the field (the samples), and we also store the interpolation method that will permit us to reconstruct the field in a computer. I also argue in the following that FieldGML offers a better representation than current ones because: (i) it takes into account the nature of datasets as found in GIS-related applications; (ii) it is valid for fields in 2D and 3D, but can be readily extended to higher-dimensions; and (iii) it is flexible in the sense that different types of fields can be stored (scattered points, tessellations, tetrahedralizations, voxels, etc.).

The report starts in Chapter 2 with definitions related to fields, then Chapters 3 and 4 give an overview and a critique of the current standards and formats used in practice, and the details of FieldGML are presented in Chapters 6 and 7.

It should be noticed that the topic was investigated from a scientific (GIS and Earth sciences) point of view, and not from the point of view of practitioners working in the industry (low-level details are therefore not discussed as the problem is approached from a conceptual point of view).

2 Fields and their Representation

Space can be conceptualised according to two different approaches: the *object* and the *field* views (Couclelis, 1992; Goodchild, 1992; Peuquet, 1984). In a nutshell, the former view considers space as being 'empty' and populated with discrete entities embedded in space. The entities can be for example roads, cups of tea, churches, etc., and they have certain properties. The latter model considers the space as being continuous, and every location in space has a certain property (there is *something* at every location). Entities or things are formed by clusters of properties. This dichotomy is also present at the implementation level for GIS applications: the raster versus vector structures.

2.1 Definition of a Field

A field is a concept rather difficult to define because it is not tangible and not part of our intuitive knowledge. It is easy for us to see and describe entities such as houses or chairs, but, although we can imagine fields, they are somewhat an abstract concept. The consequences of that are firstly that formalising a field is difficult, and secondly that many definitions exist in different disciplines (Peuquet et al., 1999). The definition usually used in a GIScience context is borrowed and adapted from physics. Physicists in the 19th century developed the concept of a *force field* to model the magnetic or the gravitational force, where a force (a vector with an orientation and a length) has a value everywhere in space, and changes from location to location. For most GIS applications, the vector assigned to each point of the Euclidean space is replaced by a scalar value, and we obtain *scalar fields* (it is assumed in the following that all fields are of that type).

Because each location in space possesses a value, a field must be represented mathematically. It is a model of the spatial variation of a given attribute a over a spatial domain, and it is modelled by a function, from $\mathbb{R}^d$ to $\mathbb{R}$ in a d-dimensional Euclidean space, mapping the location to the value of a, thus

$$a = f(location).$$

The function can theoretically have any number of independent variables (i.e. the spatial domain can have any dimensions), but in the context of geographical phenomena the function is usually bivariate (x, y) or trivariate (x, y, z). Notice that the domain can also incorporate time as an extra dimension, and thus we have $a = f(location, time)$.

2.2 Properties

Scale of Measurement. The value of the attribute a of a field can be measured according to different scales, and depending on the scale used different types of fields will

be obtained. It is important to discuss the different scales because the mathematical operations possible on fields will be different when different scales are used. For geographical data, the scale of measurement usually used is the one of Stevens (1946), who defines four scales:

Interval: the values of an attribute can be any real number $\mathbb{R}$, but the zero point on the scale is arbitrarily defined. Operations such as addition and subtraction are meaningful, but division is not. The Celsius and Fahrenheit scales are two obvious examples: 30℃ is 20℃ warmer than 10℃, but it is not three times warmer. Only the magnitude of the difference between two values is meaningful.

Ratio: the values of an attribute have all the features of interval measurement, but here the ratio between two values is meaningful. In other words, a ratio scale has a fixed zero point. Obvious examples are when temperatures are measured in Kelvin (instead of Celsius), or the amount of snow that has fallen in a certain region.

Nominal: the values of an attribute are simply labels, and they are meaningless. An example is a map of Europe where each location contains the name of the country. The only comparison that can be made between two values is if they are the same or not.

Ordinal: the values are also labels, but they can be ordered, e.g. a certain region can be categorised according to its suitability to agriculture from 1 to 5: 1 being poor, and 5 very good. Here comparisons such as 'greater than' and 'less than' can be made, but no arithmetic operations are possible.

The first two scales are considered as being *continuous*, and most fields studied in the geosciences fall into this category. Temperature, precipitation or salinity are examples because they can be measured precisely. I refer to this type of field as a *continuous field*. The last two scales are less common, but are useful in some applications (mostly social sciences). I refer to this kind of field as a *discrete field*.

Continuity. The use of the terms 'continuous' and 'discrete' can be misleading because in mathematics continuity has a slightly different meaning. The terms refer here to the scale of measurement, and not to the spatial continuity of a field. Indeed, both types of fields are spatially continuous, as they are represented by a function and there exists a value at every location in space.

2.3 Representation in Computers

The representation of a field in a computer faces many problems. Firstly, fields are continuous functions, and, by contrast, computers are discrete machines. Secondly, it should be stressed out that we never have access to a 'complete representation' of a geographical phenomenon. Indeed, to obtain information about a given phenomenon,

one must sample it, and reconstruct the field from these samples[1]. In the context of GIS-related applications (e.g. modelling of elevation, geosciences, geology, hydrology, bathymetry, etc.), this collection of samples is hindered by the fact that unlike disciplines like medicine or engineering, we seldom have direct access to the whole object (think of collecting data underground, or at sea for instance). And even if we have complete access to the object, it is often too expensive to sample the object everywhere.

In short, to represent a field in a computer (i.e. to be able to model a continuous phenomenon), we need to:

1. have a set of samples for the given fields—they are the "ground truth" of a field. The samples are usually point-based, but other forms can also exist (for instance an image obtained with remote sensing).

2. define a set of rules to obtain the values of the attribute studied, at any location. This operation is referred to as spatial interpolation.

2.4 Confusion between Objects and Fields

As highlighted by the ISO/OGC documents, there exists confusion between fields, objects, and their representations in computers. The confusion with these issues probably come from the fact that the definition of field itself changes from disciplines to disciplines, and that the issues can be seen from a philosophic, conceptual or implementation point-of-view (Peuquet et al., 1999).

Even in the GIS-related scientific literature there are many misunderstandings between spatial models, data structures, and spatial concepts. The confusion originates from the fact that object and field views of space are usually implemented in a GIS with respectively vector and raster models (Frank, 1992; Goodchild, 1992). A vector model represents individually each object with primitives such as points, lines and polygons (and polyhedra in 3D), and have hence been linked to the object view. On the other hand, raster representations are more or less synonymous with fields in the GIS community. However, as Goodchild (1992) points out, fields can also be represented by vector models (e.g. if a triangulation is used), and that is probably the main source of confusion: the same spatial model can be implemented with different data structures. According to Frank (1992), a spatial model offers an abstract view of a data structure, which is defined as "the specific implementation of a geometric data model, which fixes the storage structure, utilization, and performance".

An other example of confusion is with full partitions, for example with cadastral maps (Galton, 2001; Peuquet et al., 1999). In that case, we have objects (each parcel), and to each object is assigned one value (e.g. the owner); a field is thus present as there is a single value for the "owner" attribute at every location. The same be said of choropleth maps, which are discrete fields and used mostly in social sciences for the visualisation of

[1] Even if a sensor is used to collect samples, the result (e.g. an image with pixels) is not a complete representation since each pixel usually averages the value of the studied phenomenon over the pixel area, or each pixel represents the value located in the middle of the pixel.

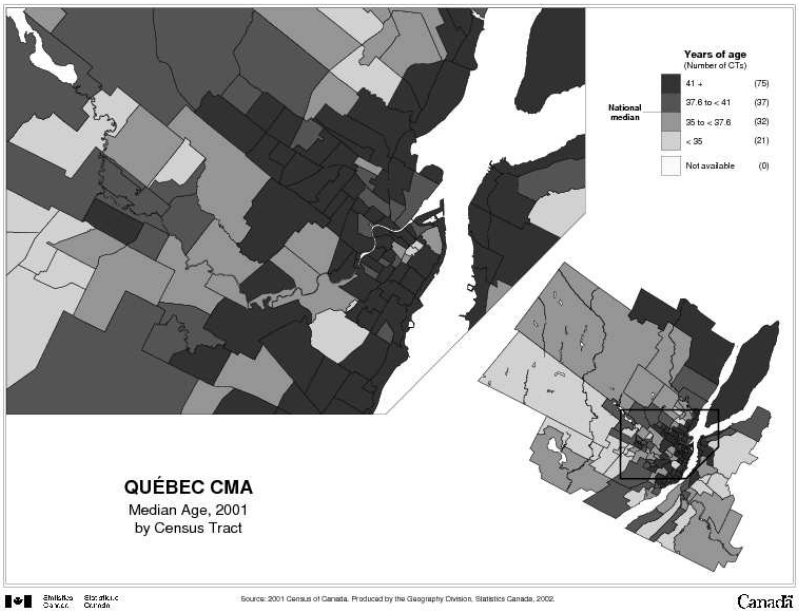

Figure 2.1: An example of a choropleth map (figure from Statistics Canada).

statistical variables such as densities, rates or proportions. Figure 2.1 shows one example where the median age of the population for different districts of Québec City is shown.

3 Current Standards for Fields

There are two "levels" of geographic information standards: abstract and implementation specifications. The former defines a conceptual architecture (or reference model) for different aspects related to the storage and exchange of information; and the latter are at a lower-level, i.e. they define an interface to access the properties and methods of classes defined in the abstract specifications.

3.1 Abstract specifications

In the case of fields, two documents exist: the 'Schema for coverage geometry and functions' (ISO, 2005b), and the OGC document with the same title (OGC, 2007b). Notice here that fields are referred to as "coverages" in these documents; both terms are synonymous and used interchangeably in the following. Both documents have the same content.

3.1.1 Definition of Coverage

A coverage is considered a *feature*[1], like is any geographic object in the ISO/OGC documents. So while each geographic object in a representation of a field is a feature, the field as a whole is a feature too.

The formal definition of "coverage" is the following (and its principal classes are shown in Figure 3.1):

> A coverage is a feature that acts as a function to return values from its range for any direct position within its spatial, temporal or spatiotemporal domain. [...] [it] has multiple values for each attribute type, where each direct position within the geometric representation of the feature has a single value for each attribute type.

Notice that a ISO/OGC coverage can have many different attribute types, but that this is not relevant here, and we simply assume that one coverage is for one attribute type (let that be the temperature of the air, the elevation of a terrain, the density of the population, etc.)

3.1.2 Two Different Types of Coverages

The coverage type is divided into two distinct but closely related subtypes:

[1] A feature is an abstraction of a real world phenomenon; it is a geographic feature if it is associated with a location relative to the Earth (ISO, 2003).

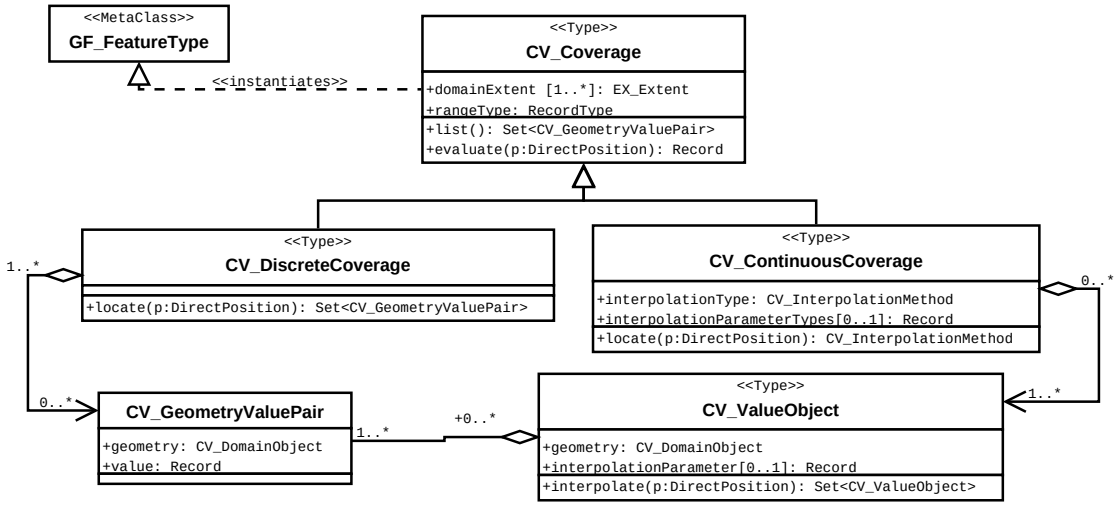

Figure 3.1: UML diagram for the main classes of an ISO/OGC coverage. (Figure after ISO (2005b))

Continuous Coverage: coverage that returns different values for the same feature attribute at different direct positions within a single spatial object, temporal object or spatiotemporal object in its domain.

Discrete Coverage: coverage that returns the same feature attribute values for every direct position within any single spatial object, temporal or spatiotemporal object in its domain.

The definition of a *continuous coverage* is more or less equivalent to that of the general coverage type. The definition refers to the fact that interpolation is used to obtain the attribute value at a given location x. The terms "within a single object" can be misleading, but means that interpolation is always performed with a function defined over one geometric object (e.g. a polygon in 2D); if no object is present at a location x (possible according to the definition of a coverage) then no value at x is returned.

The latter type, the *discrete coverage*, seems to exist only because "a coverage can be derived from a collection of discrete features with common attributes" (ISO, 2005b). As explained in Section 2.3, this is true (the samples), provided that we have a set of rules to reconstruct the coverage at every location, but this is not the case in the ISO/OGC documents. It is also stated that "a discrete coverage has a domain that consists of a finite collection of geometric objects and the direct positions contained in those geometric objects". The problem here is that these geometric objects do not have to fully partition the domain, i.e. according to that definition a set of unconnected lines and/or polygons (in which each object has a value attached to it) is considered a coverage. Even worse, the objects are permitted to overlap, which means not only do we have locations without any answer, but that there can be more than one answer at one given location! This might be useful for some applications—I am however not aware of any—but none are mentioned in the documents.

Summary. There is, in short, almost no differences between a discrete coverage and a normal vector map. That implies that a dataset composed of roads (polyline features only) to which an attribute is attached (name of the road, maximum speed, number of accident, etc.) can thus be classified as coverage (a *CV_SegmentedCurveCoverage*) or as a network of roads. Also, a continuous coverage is a discrete coverage to which an interpolation function is assigned. A set of scattered point by itself is thus a discrete coverage, but if an interpolation function is assigned to that set then it becomes a continuous coverage. A 2D grid by itself, where each pixel has a single value, is a discrete coverage (even though at every location x one can obtain a value for the attribute), and it becomes a continuous coverage if interpolation within each pixel is used (e.g. bilinear interpolation).

3.1.3 Special Coverage Subtypes

Different subtypes, which are conceptually the same as other types of coverages, are also defined and described in the ISO/OGC documents: for example *CV_TINCoverage* and *CV_ThiessenPolygonCoverage*, which are both similar since they are actually irregular partitions. Moreover, the oddity here is that these two are subtypes of *CV_ContinuousCoverage* and not *CV_DiscreteCoverage*. This is because one and only one interpolation function is assumed possible for each coverage (linear interpolation in TIN, and natural neighbour interpolation for the Voronoi diagram).

3.1.4 Interpolation Functions

The list of interpolation methods discussed in the ISO/OGC documents is very restricted. Many interpolation methods are simply ignored, and if one wanted to use them it would be very difficult to integrate them in the coverage framework. The OGC has another document discussing interpolation functions that lists more interpolation functions (OGC, 1999), but there are no formal description of these or of how they could be incorporated in the abstract specifications.

Examples of major omissions: IDW and Kriging are not listed, and as stated before, the *CV_TINCoverage* assumes for example that only one type of interpolation is possible within each triangle (which is restrictive in practice).

3.1.5 Discrete Coverages with Lines and/or Polygons & Interpolation

Usually point samples are assumed to form the basis on which fields will be reconstructed. However, the ISO/OGC also allows lines, and disconnected and/or overlapping polygons.

A set of lines, for example contour lines to model the elevation, is considered a discrete field. There is however no mention of methods to reconstruct the terrain directly from the lines (and derive a coverage). The reconstruction is usually done by discretising first the lines to points, although some theoretical work has been done for interpolation directly with points and lines (Anton et al., 2004). The only kind of interpolation mentioned for line segments is linear within each segment.

Functions to obtain continuous coverages from a set of polygons are not mentioned either, and I am not aware of any methods published that take disconnected polygons as input.

3.1.6 Three-dimensional Case

The ISO/OGC documents state at different places that the abstract specifications are valid not only for the 2D case, but also for three and higher dimensions. The problem is that it is only stated at a few places, and that the only 3D type defined is *CV_DiscreteSolidCoverage*, which is basically any datasets containing polyhedra in 3D space (these polyhedra are potentially overlapping each other and/or disconnected).

3.2 Implementation Specifications

To my knowledge, the only implementation of the ISO/OGC abstract specifications is that of GML. It is an XML-based modelling language developed to facilitate the exchange of geographic data, and has been fairly successful in recent years. While a GML file is verbose (and thus files can become enormous), there are many advantages to using it. Lake (2000) mentions, among others: (i) it is self-descriptive, (ii) it can be processed with already existing XML software, (iii) there are mechanisms to store metadata, and (iv) data integrity can be verified with the help of *schemas*. The reader is referred to Lu et al. (2007) and OGC (2007a) to learn more.

As of GML version 3.2, only the *CV_DiscreteCoverage* types have been implemented: there are GML schemas for all subtypes of *CV_DiscreteCoverage*, and also for grids (*CV_Grid*). That results in a representation that does not necessarily cover the whole spatial domain, and no mechanisms are present to estimate the value of an attribute where there are no spatial objects, or a default and simplistic method is assumed. Using simplistic interpolation methods, or the wrong parameters for a method, is dangerous as many researchers have highlighted (see Watson (1992) for instance).

Not all abstract classes were implemented in GML, *CV_GeometryValuePair* is for instance not present, and was replaced by an implementation that follows closely the conceptual distinction between the spatial domain and the range (the attribute modelled). The resulting XML file has to have three separate types: the domain, the range and another one for mapping these two correctly. As Cox (2007) explains, although this is conceptually valid, it also hinders the use of these standards in practice because of the difficulties of processing large files, of updating files, etc.

4 Summary of Formats/Models Being Used in Practice

Among GIS practitioners, fields are being used almost exclusively in 2D, while in the geoscience community 3D and higher-dimensional fields are extensively used. Note that the dimensions in oceanographic/atmospheric coverages are not necessarily spatial dimensions, as any parameters (e.g. temperature of the air, or density of water) can be considered a dimension.

As mentioned before, within the GIS community, coverages are more or less synonymous with grids, although it must be said that TINs are also widely used for modelling terrain elevation. There exist many different formats for 2D grids, but they can be easily all converted to one another.

In geoscience, netCDF[1] seems to be the *de facto* standard to exchange datasets, although other similar formats, such as HDF5[2], are also popular. These formats are raster-based, and permit users to use n-dimensional grids, with different spacing for different dimensions. They are binary and spatially structured, which means that parts of a dataset can be efficiently retrieved and processed. Although it is technically possible to store scattered points with a netCDF file (with many workarounds), the result is non-efficient and all the advantages of the format (directly access, slicing, etc.) are lost.

The use of other representations in 3D (e.g. tetrahedralizations or arbitrary polyhedra) is very rare and mostly limited to the academic community.

It should be noticed here that the wide popularity of raster representations in GIS applications is probably due to the fact that they permit us to integrate remote sensing images and fields, and also due to simplicity. Indeed, a grid is naturally stored in a computer as an array (each grid cell is addressed by its position in the array, and only the value of the grid cell is stored), and thus the spatial relationships between cells are implicit. The algorithms to analyse and manipulate (boolean operations such as intersection or containment) are also trivially implemented in a computer.

4.1 The Dangers of Using Raster Formats

As argued by many over the years, using raster structures has many drawbacks (Gold and Edwards, 1992; Kemp, 1993; Haklay, 2004; Ledoux and Gold, 2006). Firstly, as Fisher (1997) points out, the use of pixels as the main element for storing and analysing geographical data is not optimal. The problems most often cited are: (i) the meaning

[1] `http://www.unidata.ucar.edu/software/netcdf/`
[2] http://www.hdfgroup.com

of a grid is unclear (are the values at the centre of each pixel, or at the intersections of grid lines?), (ii) the size of a grid (for fine resolutions, grids can become huge), (iii) the fact that the space is arbitrarily tessellated without taking into consideration the objects embedded in that space. Secondly, the use of grids in GIS/geoscience applications has wider repercussions since we can assume in most cases that a given grid was constructed from a set of point samples. Converting samples to grids is dangerous because the original samples, which could be meaningful points such as the summits, valleys or ridges or a terrain, are not present in the resulting grid. The importance of the original samples for a field is such that they have even been dubbed the *meta-field* by Kemp and Včkovski (1998). It should also be said that when a user only has access to a grid, he often does not know how it was constructed and what interpolation method was used, unless metadata are available. Notice that all the previous statements are also valid in 3D (a pixel becomes a voxel).

5 Other Efforts to Represent Fields

From a "standards" point of view, different XML-based languages (eXtensible Markup Language) have been proposed. First of all, there is the more general-purpose GML that implements many of the ISO/OGC standards for fields, but not all of them. Note that the definitions of these standards can be found in Section 3, and their implementation with GML is discussed in Section 6. Based on GML/XML, there are different languages to model fields. For instance, Nativi et al. (2005) propose the *NcML-GML*, which permits us to store with GML the metadata associated with netCDF files (this is a multi-dimensional raster format described in the next section). Also, Woolf and Lowe (2007) propose the *Climate Science Modelling Language* (CSML), which is used to represent all the different kinds of climate data (often fields) and their relevant information. The particularity of CSML is that, for the sake of simplicity and performance, the authors chose to use only parts of the standards: they offer a GML-based 'wrapper' around legacy formats to simplify exchange, but they are still using the legacy file for applications (these legacy files are all raster-based). Furthermore, the *Geoscience Markup Language* (GeoSciML) can be used to store any kind of information related to geology (Sen and Duffy, 2005). When fields are involved, they are usually stored in raster formats, but GeoSciML also allows the storage of the observations that were collected (interpolation methods are however not discussed).

From a GIScience point of view, different alternatives to the ubiquitous rasters have been proposed over the years, starting with tessellations into triangles (Mark, 1975; Peucker, 1978). Kemp (1993) proposes different alternatives to store 2D fields, and shows how to convert them from one representation to another when needed (for analysis). Gold and Edwards (1992), and Ledoux and Gold (2006), among others, have also discussed the use of the Voronoi diagram (in 2D and 3D) as an interesting alternative to raster-based approaches. In a proposition that is similar to the one in this paper (at least conceptually), Haklay (2004) proposes, in an attempt to model and manipulate 2D fields, to store only the samples collected, and the parameters of interpolation functions.

FieldGML is also conceptually very similar to the concept of *virtual data set* (VDS) (Stephan et al., 1993; Včkovski and Bucher, 1996; Včkovski, 1998). A VDS is a dataset "enhanced" with a set of methods that are used to access, manipulate or transform the data—it is an object in the object-oriented sense of the term. The term "virtual" means that different representations of a dataset can be generated for different users/applications. In the context of fields, that means that the samples of a field are stored, and also that interpolation methods to generate different representations of that field are available (pixel size, format, data model, etc.). It is implemented as a Java class where an interface is defined.

VDSs were introduced around 15 years ago as a solution to the interoperability of GISs

and to improve the quality of datasets used in GIS. The whole concept of interoperability through VDS was based on the idea that "data exchange is not specified by a standardised data structure (e.g. a physical file format) but a set of interfaces" (Včkovski, 1998, p.54). If we fast-forward to 2008, we now have widely-accepted GIS-related standards (see Section 3) and even a *de facto* language (GML). These standards have taken a different approach to interoperability since all datasets are coded with the same language, which clearly contrasts with VDS where one could store the datasets in his own format as long as he/she implemented the interface. FieldGML can thus be seen as implementation of the conceptual ideas of VDS in a 2008 context where GML is synonymous with interoperability in the GIS world.

6 FieldGML: The Field Geography Markup Language

The previous chapters have highlighted that while the current abstract standards for coverages do avoid the distinction between raster and vector, the two types created (discrete and continuous coverages) are problematic because the distinction between them is rather blurred and subtle, and no implementation exist for the latter type. The current standards contributes to the confusion that already exists among users about fields.

Because of these shortcomings and weaknesses, I propose an alternative to represent fields: FieldGML. It is an XML-based language based on GML, and it permits us to represent fields in 2D and 3D, although conceptually it can be easily extended to higher dimensions. Unlike current standards where there is a distinction between discrete and continuous fields/coverages, I argue in this paper that a field should always have one—and only one!—value for a given attribute at every location in the spatial domain (be this domain the surface of the Earth, a 3D volume, or even a 4D spatio-temporal hypercube). The concept of discrete coverage can be then removed, as it is misleading and confusing.

6.1 A Field = Samples + Interpolation Rules

The principal idea behind FieldGML is that two things are needed to have a coverage:

1. a set of samples of the phenomenon;
2. an interpolation function to reconstruct the continuity of the phenomenon studied.

Samples. By that it is meant what is referred to as 'discrete coverage' in ISO/OGC terms. It is any data that were collected to study the phenomenon:

1. a set of scattered points in 2D or 3D.
2. a set of lines, e.g. contour lines coming from a topographic map.
3. a set of scattered polygons to which one value is attached. Although this case is possible, I am not aware of any interpolation method that would take a set of polygons as input. It is nevertheless always possible to discretise each polygon into a set of points. Polyhedra in 3D are also considered samples.
4. a raster image coming from remote sensing or photogrammetry where the value of each pixel represents the temperature of the sea for instance.

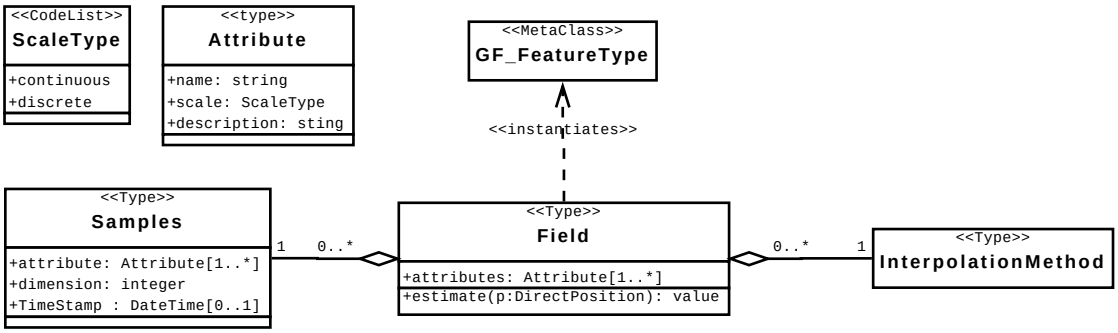

Figure 6.1: Overview of FieldGML classes.

Observe that a set of samples is simply a "normal" vector file or a grid (as defined in other ISO/OGC standards, e.g. in ISO (2003)), where each object is assigned a value for a common attribute.

Interpolation Method. The set of rules used to reconstruct the field from samples can take many forms. Interpolation methods are rather difficult to categorise because they are based on different paradigms, and some methods fall into more than one category. No attempts will be made here to introduce categories (see Mitas and Mitasova (1999) and Watson (1992) for that), but what should be kept in mind is that although several different interpolation methods are used in GIS and that several publications advocate the use of "better" methods, the current standards, while discussing a few methods, give no importance to interpolation and do not permit the use of many of the known methods.

Storing explicitly the interpolation method, as FieldGML is doing, is efficient in practice as only a few parameters have to be stored. Finding the appropriate values for interpolation parameters is a difficult and time-consuming task, as the user must have a good understanding of the spatial distribution of the objects in the set of samples, and of the details of the method. A vivid example is Kriging (Oliver and Webster, 1990), with which experienced users can obtain very good results, but which also leaves newcomers clueless with its many parameters and options. Using Kriging with the appropriate parameters leads to a result that has statistically minimum variance, however, simply using the default values for the parameters will most likely lead to unreliable results. Thus, if we leave the job of modelling the datasets and deriving the interpolation parameters to specialists, the users would not have to worry about these anymore. This is one of FieldGML's main benefits.

6.2 Abstract Specifications

Figure 6.1 shows an overview of the classes of FieldGML. The first thing to notice is that the FieldGML type *Field* instantiates metaclass GF_FeatureType (of ISO19109 (ISO, 2005a)), which means a field is a feature. A field contains one set of samples (*Sample*

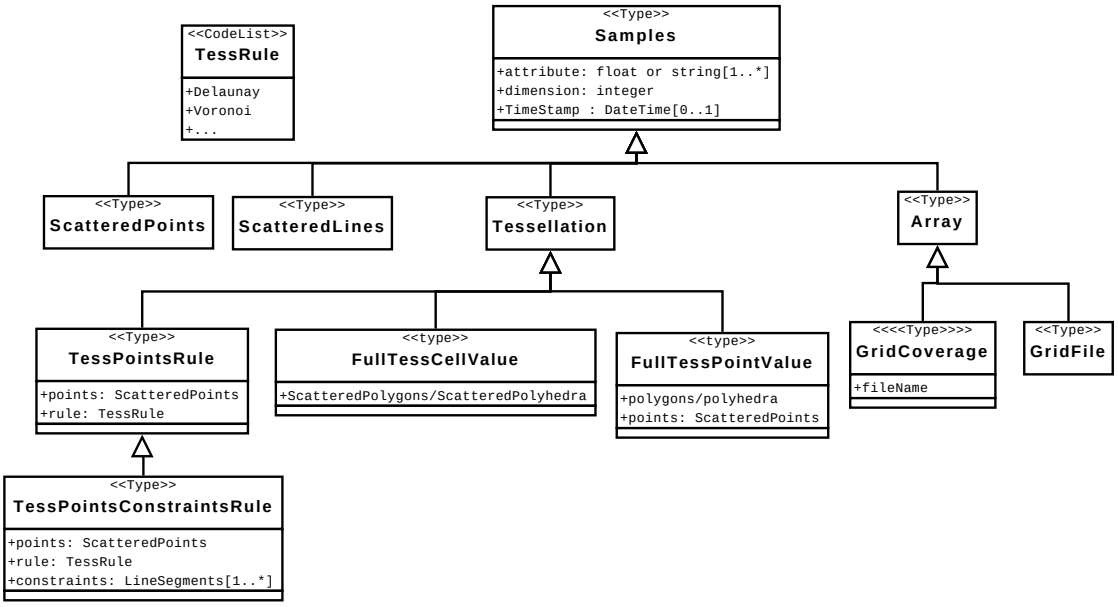

Figure 6.2: The Sample type.

type) and one interpolation method (of type *InterpolationMethod*. A field can model more than one attribute, and the value assigned is of type *Attribute*. A field contains one major operation, *estimate*, which takes as input a DirectPosition (a location in 2D or 3D, depending on the field) and an attribute, and outputs the value of the modelled attribute. This value is obtained by spatial interpolation, using the *InterpolationMethod* class.

6.2.1 Samples

The Samples type, as shown in Figure 6.2 is the parent class for many different kinds of samples that can be collected to study a field. It contains three attributes:

- **attribute:** that contains the name of the attribute, the scale of measurement (either continuous or discrete) and a description.
- **dimension:** that can be either "2" or "3", or theoretically a higher value.
- **TimeStamp:** as explained below, it is possible to time-stamp a collected dataset. A field can therefore be a dynamic field.

The sub-classes of Samples are the following:

- **ScatteredPoints:** contains a set of points, regularly or irregularly distributed in space. The dimension of each point can be either 2D or 3D.

- **ScatteredLines:** contains a set of lines, regularly or irregularly distributed in space. The dimension of each point can be either 2D or 3D, but in practice that will most likely be 2D and the lines iso-contours.

- **Tesselation:** A tessellation is a partition of the space by a set of disjoint elements, such that there are no empty parts (the union of all the elements completely fills the space). The elements can be 2D (triangles or polygons), or 3D (tetrahedra or polyhedra).

- **Array:** As is the case for GeoSciML (Woolf and Lowe, 2007), it was decided that 'legacy files' (i.e. raster formats used in commercial GISs) could be used directly without having to convert them to GML types (which are non-efficient and cumbersome to use in practice). It is however still possible to use *CV_Grid* as defined in ISO/OGC standards and implemented in GML (as a discrete coverage). Legacy files are simply referenced to by a pointer; the metadata about the file (georeferencing, pixel size, etc.) have however to be stored in the FieldGML file with GML types and/or attributes.

Types of tessellations. The Tessellation type is further divided into four sub-classes:

- **TessPointsRule:** That class contains a ScatteredPoints, plus the rule to be used to construct the tessellation. The idea is not to store explicitly all the elements of the tessellation, and reconstruct them on-the-fly when needed. The two best examples are the Delaunay triangle and the Voronoi diagram of a set of points, but any other tessellation could be used, as long as it can be created automatically (one can think of the different generalisations of Voronoi diagrams).

- **TessPointsConstraintsRule:** When creating a tessellation, it is also possible to consider constraints, which are usually line segments (in 2D, but polygonal faces in 3D are also possible). The most common example is the Constrained Delaunay triangulation, see Seidel (1988) for more info.

- **FullTessCellValue:** This is a tessellation formed by a set of polygons or polyhedra, where each element has a value. In 2D, that is choropleth map for instance. The types ScatteredPolygons and ScatteredPolyhedra, could be used, as long as they form a valid tessellation. This type will in practice mostly used for discrete fields.

- **FullTessPointValue:** This type exist because in a tessellation, the value is not always attached to the elements of higher dimensionality. There can be tessellation where the values are attached to the points forming the element. One example is a TIN, where each triangle does not have one value since these are attached to the points. The type FullTessPointValue exists mostly because not all triangulation can be automatically reconstructed, for instance if it was manually constructed.

Scale of measurement. The scale of measurement for a field is important because the mathematical operations possible on a field will be different when different scales are used. For instance, a field whose samples have an attribute with a discrete scale cannot be reconstructed with most of the interpolation methods. If the values are attached to points, then only NearestNeighbour should be used, and if they are attached to a polygon for instance, then a constant function inside each polygon should be used. Also, while many arithmetic operations (addition, subtraction, multiplication, etc.) are possible on continuous fields, they are meaningless for discrete fields.

Dynamic field. In the ISO/OGC documents, there is no meaningful discussion concerning the temporal aspect, i.e. how dynamic fields should be modelled. It is unclear at this moment whether time should be treated as another dimension or whether it should be treated separately; these two cases are respectively called "integrated" and "hybrid" cases (Galton, 2004; Raper, 2000). A full integration would mean that space and time exhibit the same characteristics, and that spatio-temporal hyper-objects are created in the database. That would also mean that interpolation is performed in for example 4D in the case of datasets referenced in 3D space. Such interpolation methods are technically possible, but finding a distance function that is valid both in space and time is tricky, and very much application-related (Zhang and Hunter, 2000). As for the hybrid case, it means that space and time and treated separately, so that for instance interpolation for temporal data is always performed in space, and then in time. Assume we have different datasets collected at different times t_{0h}, t_{5h} and t_{8h}. To know what the situation was at $t = 2h$, one would therefore use interpolation in space at $t = 0h$ and at $t = 5h$, and then interpolate in time between the two values obtained. FieldGML permits users to time-stamp either a Samples class (so that all the samples in it have the same time stamp), or to time-stamp each elements of a dataset. That way, the user can decide which interpolation method and how to implement it (integrated or hybrid model).

6.2.2 InterpolationMethod

Interpolation methods play an important role in the FieldGML model, and several methods used in the GIS world have been listed. The list of methods in Figure 6.3 is by no means exhaustive as other ones can be easily be added if needed (which is a big advantage over current standards). It should also be noticed that all the methods listed are perfectly valid in 2D and 3D.

A few examples of the methods commonly used in GIS:

Piecewise: a function is defined over each cell of a tessellation. The function within each element is usually a simple mathematical function, and as Goodchild (1992) points out, this function can be constant, linear, or of a higher order. A constant function means that the value of the attribute is constant within one region, for example to represent a discrete field, as in a choropleth map. An example of the use of a linear function is a TIN: the spatial variation within each region (a triangle) is

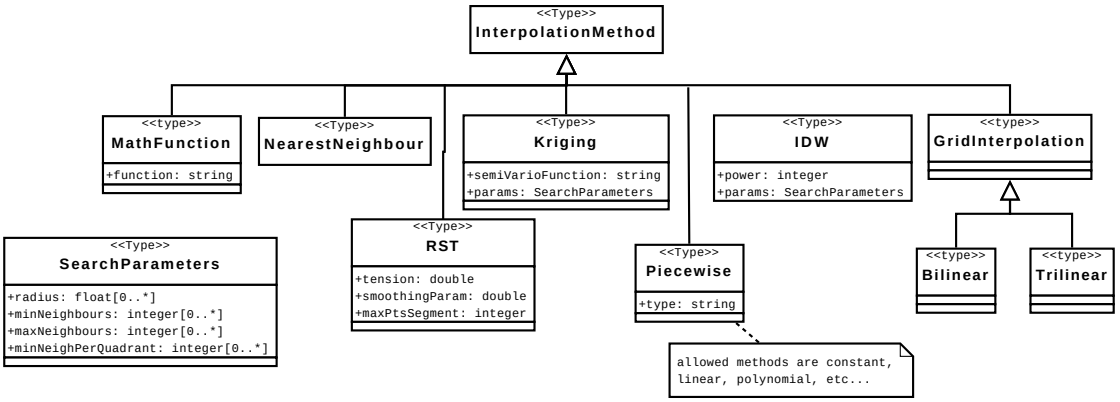

Figure 6.3: The InterpolationMethod type.

described by the linear function (a plane) defined by the three vertices lifted to their respective elevation. The value of the attribute of a field at an unsampled location x is thus obtained by linearly interpolating on the plane passing through the three vertices of the triangle containing x. Akima (1978) shows the advantages of using higher order functions in each region of a TIN—the main one being that the slope of the terrain is continuous everywhere, i.e. there are no discontinuities at the border of two triangles. The same functions can obviously be used within each element in three dimensions.

IDW: as described in Shepard (1968), it requires different parameters to define which points are involved in the interpolation at a given location (different criteria can be used), and also the power must be defined.

Kriging: while the modelling of a dataset is a difficult and time-consuming task, the output of the modelling (a function characterising the dependence between the attributes of any two samples that are at a given distance from each other) can be simply stored as a string. The parameters and the functions as defined in the program *gstat* (Pebesma and Wesseling, 1998) are used.

Natural neighbour: the basic method (Sibson, 1981) does not need any user-defined parameters, but it is possible to obtain a smoother interpolation if certain parameters are used (see Watson (1992)).

RST—regularized spline with tension: this method is available in the open-source GIS GRASS, and by storing a few parameters a field can be reconstructed from a set of samples (Mitasova and Mitas, 1993).

Grid interpolation: while a grid can be seen as a special case of scattered points, different methods optimised for grids have been developed, for instance bilinear and biquadratic. See Kidner et al. (1999) for a discussion in 2D, but these methods trivially generalise to higher dimensions.

6.3 Implementation Specifications

The abstract specifications were implemented as an GML application schema. The application schema is not fully described here since the classes of the abstract specifications all become straightforwardly XML types. The schema, `fieldgml.xsd`, can be obtained and browsed on the website of FieldGML[1].

What follows is an overview of the important engineering decisions that were taken in order to develop FieldGML and its schema. I tried to use GML types as much as possible, but for practical reasons (e.g. simplicity of implementation and performances for processing files) several new types also had to be created. An important decision that was taken was not to use directly the GML implementation of *CV_DiscreteCoverage* (called `gml:DiscreteCoverage`) for the set of samples, for the reasons described previously.

Field type. A Field is a GML feature (of the type `gml:AbstractFeatureType`, which means that it can use all the mechanisms already defined by the OGC to deal with metadata.

```
<xs:element name="Field" type="FieldType"/>
<xs:complexType name="FieldType">
 <xs:complexContent>
  <xs:extension base="gml:AbstractFeatureType">
   <xs:sequence>
    <xs:element name="Samples" type="SamplesType"/>
    <xs:element type="InterpolationMethodType" name="InterpolationMethod"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Samples type. It inherits, and all his children, from `gml:AbstractGeometryType`, which means that mechanisms defined by GML for reference systems can be used.

```
<xs:complexType name="SamplesType">
 <xs:complexContent>
  <xs:extension base="gml:AbstractGeometryType">
   <xs:sequence>
    <xs:element name="Attribute" type="AttributeType" minOccurs="1"
      maxOccurs="unbounded"/>
    <xs:element ref="gml:TimeInstant" maxOccurs="1" minOccurs="0"/>
    <xs:choice>
     <xs:element ref="ScatteredPoints"/>
     <xs:element ref="ScatteredLines"/>
     <xs:element type="ArrayType" name="Array"/>
     <xs:element name="Tessellation" type="TessellationType"/>
    </xs:choice>
   </xs:sequence>
```

[1] `www.gdmc.nl/ledoux/fieldgml.html`

```
   <xs:attribute name="dimension" type="NumDimensionType" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

CV_GeometryPairValue. The sub-types of Samples were all extended so that an attribute (Attribute type) is attached to each object. A version of the *CV_GeometryPairValue* was implemented, as in Cox (2007). For instance, for the ScatteredPoints, PointValue was also created, it is an extension of `gml:PointType`

```
<xs:complexType name="AttributeType">
 <xs:sequence>
  <xs:element name="Name" type="xs:string"/>
  <xs:element name="Scale" type="AttributeScaleType"/>
  <xs:element name="Description" type="xs:string" minOccurs="0" maxOccurs="1"/>
 </xs:sequence>
</xs:complexType>

<xs:element name="PointValue" type="PointValueType" substitutionGroup="gml:Point"/>
<xs:complexType name="PointValueType">
 <xs:complexContent>
  <xs:extension base="gml:PointType">
   <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element name="value" type="xs:anyType"/>
    <xs:element ref="gml:TimeInstant" maxOccurs="1" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:element name="ScatteredPoints" type="ScatteredPointsType"
 substitutionGroup="gml:_GeometricAggregate"/>
<xs:complexType name="ScatteredPointsType">
 <xs:complexContent>
  <xs:extension base="gml:AbstractGeometricAggregateType">
   <xs:sequence>
    <xs:element ref="PointValue" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

Time stamping. To time-stamp elements, the `gml:TimeInstant` type was used.

TessPointsRule. It should be noticed here that if a tessellation is needed for the interpolation (e.g. Delaunay triangulation for a piecewise interpolation) this structure need not be persistent: only the samples can be stored, and it is calculated on the fly by the programme handling FieldGML files (see Chapter 7.

```
<xs:simpleType name="TessRuleType">
 <xs:restriction base="xs:string">
  <xs:enumeration value="Delaunay"/>
  <xs:enumeration value="Voronoi"/>
 </xs:restriction>
</xs:simpleType>

<xs:complexType name="TessPointsRuleType">
 <xs:sequence>
  <xs:element ref="ScatteredPoints"/>
  <xs:element type="TessRuleType" name="TessRule"/>
 </xs:sequence>
</xs:complexType>
```

FullTessCellValue. This type is simply a composite of `PolygonValue`, which is a polygon with a value attached to it. The types `gml:_Surface` were not used, because it would have required the creation of several new types. Here, the user is responsible for insuring that the polygons (or polyhedra in 3D) form a valid tessellation.

```
<xs:element name="FullTessPolygonValue" type="CompositePolygonValueType"
  substitutionGroup="gml:_GeometricAggregate"/>
<xs:complexType name="CompositePolygonValueType">
 <xs:complexContent>
  <xs:extension base="gml:AbstractGeometricAggregateType">
   <xs:sequence>
    <xs:element ref="PolygonValue" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

FullTessPointsValue. The solution retained is rather simplistic, but representing this concept with the existing GML types would require a lot of efforts. The reason is that when polygons or solids are involved, `gml:Point` types are not used, only `gml:posList`, which would be cumbersome to extend with an Attribute value (a `gml:Polygon` can easily be extended with an Attribute value, but we want here to have values attached to the points). Therefore, two elements must be present (for the 2D case): (i) a "normal" `gml:CompositeSurface`; and (ii) a `ScatteredPoints` (which contains all the points of the surface, with the Attribute values). This solution has the side benefit that every value for a point will be stored only once, and not once for each polygon (since `gml:CompositeSurface` is not "topologic").

```
<xs:complexType name="FullTessPointValue2DType">
 <xs:sequence>
  <xs:element ref="gml:CompositeSurface"/>
  <xs:element ref="ScatteredPoints" maxOccurs="1" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
```

Interpolation methods. Storing the interpolation methods is very simple, some methods do not even require any parameters (for instance nearest neighbour interpolation (Sibson, 1981).

```
<xs:complexType name="InterpolationMethodType">
 <xs:choice>
  <xs:element name="Kriging" type="KrigingType"/>
  <xs:element name="NearestNeighbour" type="NearestNeighbourType"/>
  <xs:element name="IDW" type="IDWType"/>
  <xs:element name="NaturalNeighbour" type="NaturalNeighbourType"/>
  <xs:element name="RST" type="RSTType"/>
  <xs:element name="Piecewise" type="PiecewiseType"/>
  <xs:element name="MathFunction" type="xs:string"/>
  <xs:element name="Bilinear" type="BilinearType"/>
  <xs:element name="Trilinear" type="TrilinearType"/>
 </xs:choice>
</xs:complexType>

<xs:complexType name="IDWType">
 <xs:sequence>
  <xs:element name="Power" type="xs:positiveInteger"/>
  <xs:element name="SearchParam" type="SearchParamType"/>
 </xs:sequence>
</xs:complexType>

<xs:complexType name="SearchParamType">
 <xs:sequence>
  <xs:element name="Radius" type="xs:double" maxOccurs="1" minOccurs="0"/>
  <xs:element name="MinNeighbours" type="xs:positiveInteger" maxOccurs="1"
    minOccurs="0"/>
  <xs:element name="MaxNeighbours" maxOccurs="1" minOccurs="0"
    type="xs:positiveInteger"/>
  <xs:element name="MinNeighboursQuadrant" type="xs:positiveInteger"
    maxOccurs="1" minOccurs="0"/>
 </xs:sequence>
</xs:complexType>
```

6.4 Summary

It should be highlighted here that if fields are represented with FieldGML, any kind of fields can be defined, and special cases used by ISO/OGC all fall into one category (so there are no needs to define explicitly subtypes). For example:

- the *CV_TINCoverage* is based on a set of points, and the interpolation is a piecewise function (linear function inside each Delaunay triangle).
- the *CV_ThiessenPolygonCoverage* type is obtained with the set of scattered point that was used to create the Voronoi/Thiessen polygons, and the interpolation method is natural neighbour; notice that if a constant function is assigned to each

polygon, then we obtain the same results as if the nearest neighbour interpolation was used.

- the *CV_DiscreteGridPointCoverage* type is obtained with a GridFile for instance. Observe here that even if a grid is the set of samples for a field, an interpolation method must be also defined (it can be for instance constant or bilinear inside each cell).

7 Prototype and Examples of FieldGML files

To convert back and forth between the FieldGML representation and the formats used in GIS and geoscience applications (mostly grids), a prototype was built. The prototype was developed with the Python programming language, and uses only open-source software.

Currently it permits users to read a FieldGML file and output to different formats, and it is also possible to create a FieldGML file when a set of samples is already available. To output to a format used by commercial GISs, the user has to choose the resolution of the grids (only grids are possible right now, although triangulation could be implemented in the future), and the format. The possible grid formats currently supported are the ones in the GDAL library[1] (in 2D), and netCDF (in 3D).

Some of the interpolation methods described in the precedent section were implemented or their libraries were linked to the prototype. For instance, the program *gstat* (Pebesma and Wesseling, 1998) was used for Kriging, and CGAL[2] to create triangulations in 2D and 3D, used for some of the interpolation methods.

7.1 FieldGML files

The Appendix A contains some excerpts from FieldGML files. The complete files can be found on the FieldGML website.

7.2 fgmlinfo: to get information about a FieldGML file

This simple script gives information about a FieldGML file. Its use is simple:

```
$ python fgmlinfo 5.xml
```

and the user gets an overview of the content of a the file *5.xml*.

7.3 fgml2grid: to convert a FieldGML file to a grid

With this script, one can convert a FieldGML file to a grid, either in 2D or 3D. Right now, only *GeoTiff*, *HFA* and *netCDF* formats are supported. The user must specify

[1] The Geospatial Data Abstraction Library: `www.gdal.org`
[2] The computational geometry algorithms library: `www.cgal.org`

the format, and the pixel resolution (these have to be the same in all dimensions). For instance, for a give file *hill.xml*

```
$ python fgml2grid.py hill.xml elevation GTiff 5
```

would create a GeoTiff file (the bounding box of the dataset is used), with a 5 unit size for every pixel. The parameter "elevation" is used since we want to use this attribute (a FieldGML file can store more than one attribute at the same time).

7.4 file2fgml: to convert a dataset to a FieldGML file

This Python script permits us to convert scattered points (in 2D and 3D), and lines in 2D to a FieldGML file. The original datasets can be either in ASCII format, or in an ESRI's shapefile (for the 2D datasets). For instance, to convert a set of points representing elevation, for instance an ASCII file *5.txt*

```
$ python file2fgml.py 5.txt elevation
```

The user can then open the XML file and fill out the details for the interpolation method.

7.5 Discussion Over the Implementation

At this moment, some of the interpolation methods have been implemented in the prototype as a proof of concept, but to favour interoperability, a better option would be having a webserver where these functions are available. The newly adopted OGC standards about web processing service (WPS) (OGC, 2007c), which defines how GIS operations can be performed over the Internet could be used for instance. The interpolation methods used by FieldGML would simply be available on a server, and a user would upload his FieldGML file, specify what representation is needed, and then he/she would get the file.

Also, it is interesting to observe that while FieldGML and VDS (as described in Section 5 have very similar conceptual ideas, the implementations are totally different because of the way interoperability is tackled. The VDS approach was about having proprietary formats not directly accessible to users, who had to access data through common interfaces. While theoretically very sound, this was not the choice the GIS community picked, and now instead we have one language (GML) that can be used to represent any geographical dataset. While probably less efficient (GML is very verbose and complex), it offers more flexibility as anyone can read a FieldGML file and extract the original samples, while in the case of VDS you would have to have a piece of software implementing the interface. With the original dataset, the user can then choose another interpolation method, if needed.

8 Conclusions

The ultimate goal of a digital field representation is to reconstruct in a computer the continuity of a studied phenomenon, i.e. to be able to accurately estimate, or calculate, the value of the phenomenon at any location in a spatial domain. To do so, the current ISO/OGC standards offer two abstract types: the discrete and the continuous coverages. The main problems with these are:

- The former does not permit us to represent fields, as any GIS datasets to which an attribute is attached fall in the discrete coverage category.
- The distinction between the two types are blurred and subtle, and that has created confusion among users. As highlighted in this report, a field is somewhat difficult to represent in a computer, so the related standards ought to be clear on definitions and related concepts.
- Interpolation methods are barely discussed, and for many types only one method is assumed to be used. This is too restrictive in practice, since different methods are used in different disciplines.
- There are no known implementations of the continuous coverage type. This is probably a direct consequence of the poor definitions.

FieldGML: the alternative. As an alternative to current standards for fields, FieldGML has been proposed. With it, a field always contains two parts:

1. the set of samples that were collected to study the field;
2. the interpolation method—with values for all the parameters–that should be used to reconstruct the field in a computer.

It has the following advantages:

- it respects the scientific definition of a field.
- it is simple from a theoretical point of view, and thus easy to understand for users. A field is always something continuous; if you only have a dataset of scattered points, this is not a field.
- it permits us to model every situation (and that in two and three dimensions). Thus, no sub-types are necessary.
- it uses the types already defined in current implementation standards (i.e. GML).

- it is extensible. Users can "plug" their own interpolation methods.
- more importantly, it is more adapted than raster structures to the kind of datasets found in GIS-related applications, because it permits us to always keep the original data that were collected to study a phenomenon, and simply generate new representations that are adapted to a particular use and application.
- it is implementable. As a proof of concepts, a GML application schema was created, and some FieldGML datasets created.

Finally, it should be said that while the use of FieldGML requires a rethinking from people who produce fields, the users need not be affected. Indeed, a potential user of FieldGML would simply obtain a field in the form of a FieldGML file, select the format and resolution of the output file, and carry on with his/her work as before. But when he/she would need to exchange the field with someone else, the shortcomings of raster structures would not arise.

For more information about FieldGML:

`http://www.gdmc.nl/ledoux/fieldgml.html`

Bibliography

H. Akima. A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points. *ACM Transactions on Mathematical Software*, 4(2):148–159, 1978.

F. Anton, D. Mioc, and C. M. Gold. Line Voronoi based interpolation and application to digital terrain modelling. In *ISPRS 2004—XXth Congress*, volume II, Istanbul, Turkey, 2004.

H. Couclelis. People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS. In A. U. Frank, I. Campari, and U. Formentini, editors, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, volume 639 of *Lecture Notes in Computer Science*, pages 65–77. Springer-Verlag, 1992.

S. Cox. GML encoding of discrete coverages (interleaved pattern). Open Geospatial Consortium inc., 2007. Document 06-188r1, version 0.2.0.

P. F. Fisher. The pixel: A snare and a delusion. *International Journal of Remote Sensing*, 18(3):679–685, 1997.

A. U. Frank. Spatial concepts, geometric data models, and geometric data structures. *Computers & Geosciences*, 18(4):409–417, 1992.

A. Galton. A formal theory of objects and fields. In D. R. Montello, editor, *Spatial Information Theory: Foundations of Geographic Information Science (Proceedings of International Conference COSIT 2001)*, volume 2205 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2001.

A. Galton. Fields and objects in space, time, and space-time. *Spatial Cognition and Computation*, 4(1):39–68, 2004.

C. M. Gold and G. Edwards. The Voronoi spatial model: Two- and three-dimensional applications in image analysis. *ITC Journal*, 1:11–19, 1992.

M. F. Goodchild. Geographical data modeling. *Computers & Geosciences*, 18(4):401–408, 1992.

M. Haklay. Map Calculus in GIS: A proposal and demonstration. *International Journal of Geographical Information Science*, 18(2):107–125, 2004.

ISO. ISO 19107: Geographic information—Spatial schema. International Organization for Standardization, 2003.

ISO. ISO 19109: Geographic information—Rules for application schema. International Organization for Standardization, 2005a.

ISO. ISO 19123: Geographic information—Schema for coverage geometry and functions. International Organization for Standardization, 2005b.

K. K. Kemp. Environmental modeling with GIS: A strategy for dealing with spatial continuity. Technical Report 93-3, National Center for Geographic Information and Analysis, University of California, Santa Barbara, USA, 1993.

K. K. Kemp and A. Včkovski. Towards an ontology of fields. In *Proceedings 3rd International Conference on GeoComputation*, Bristol, UK, 1998.

D. Kidner, M. Dorey, and D. Smith. What's the point? interpolation and extrapolation with a regular grid DEM. In *Proceedings 4th International Conference on GeoComputation*, Mary Washington College Fredericksburg, Virginia, USA, 1999.

R. Lake. Introduction to GML: Geography markup language. In *Proceedings W3C Workshop on Position Dependent Information Services*, Sophia Antipolis, France, 2000. Available at `http://www.w3.org/Mobile/posdep/GMLIntroduction.html`.

H. Ledoux and C. M. Gold. A Voronoi-based map algebra. In A. Reidl, W. Kainz, and G. Elmes, editors, *Progress in Spatial Data Handling—12th International Symposium on Spatial Data Handling*, pages 117–131. Springer, 2006.

C.-T. Lu, R. F. Dos Santos, L. N. Sripada, and Y. Kou. Advances in GML for geospatial applications. *GeoInformatica*, 11:131–157, 2007.

D. M. Mark. Computer analysis of topography: A comparison of terrain storage methods. *Geografiska Annaler*, 57A(3–4):179–188, 1975.

L. Mitas and H. Mitasova. Spatial interpolation. In P. A. Longley, M. F. Goodchild, D. J. Maguire, and D. W. Rhind, editors, *Geographical Information Systems*, pages 481–492. John Wiley & Sons, second edition, 1999.

H. Mitasova and L. Mitas. Interpolation by regularized spline with tension: I. Theory and implementation. *Mathematical Geology*, 25:641–655, 1993.

S. Nativi, J. Caron, E. Davies, and B. Domenico. Design and implementation of netCDF markup language (NcML) and its GML-based extension (NcML-GML). *Computers & Geosciences*, 31(9):1104–1118, 2005.

OGC. Geography markup language (GML) encoding standard. Open Geospatial Consortium inc., 2007a. Document 07-036, version 3.2.1.

OGC. Topic 6: Schema for coverage geometry and functions. Open Geospatial Consortium inc., 2007b. Document 07-011, version 7.0.

OGC. Topic 4: Stored functions and interpolation. Open Geospatial Consortium inc., 1999. Document 09-104.

OGC. Web processing service. Open Geospatial Consortium inc., 2007c. Document 05-007r7, version 1.0.0.

M. A. Oliver and R. Webster. Kriging: A method of interpolation for geographical information systems. *International Journal of Geographical Information Systems*, 4: 313–332, 1990.

E. J. Pebesma and C. G. Wesseling. Gstat: a program for geostatistical modelling, prediction and simulation. *Computers & Geosciences*, 24(1):17–31, 1998.

T. K. Peucker. Data structures for digital terrain models: Discussion and comparison. In *Harvard Papers on Geographic Information Systems*. Harvard University Press, 1978.

D. J. Peuquet. A conceptual framework and comparison of spatial data models. *Cartographica*, 21(4):66–113, 1984.

D. J. Peuquet, B. Smith, and B. Brogaard. The ontology of fields: Report of a specialist meeting held under the auspices of the VARENIUS project. Technical report, National Center for Geographic Information and Analysis, Santa Barbara, USA, 1999.

J. Raper. *Multidimensional geographic information science*. Taylor & Francis, London, 2000.

R. Seidel. Constrained Delaunay triangulations and Voronoi diagrams with obstacles. Technical report, Institute for Information Processing, Graz, Austria, 1988.

M. Sen and T. Duffy. GeoSciML: Development of a generic geoscience markup language. *Computers & Geosciences*, 31(9):1095–1103, 2005.

D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings 23rd ACM National Conference*, pages 517—524, 1968.

R. Sibson. A brief description of natural neighbour interpolation. In V. Barnett, editor, *Interpreting Multivariate Data*, pages 21–36. Wiley, New York, USA, 1981.

E.-M. Stephan, A. Včkovski, and F. Bucher. Virtual Data Set: An approach for the integration of incompatible data. In *Proceedings AutoCarto 11 Conference*, pages 93–102, Minneapolis, USA, 1993.

S. S. Stevens. On the theory of scales and measurement. *Science*, 103:677–680, 1946.

A. Včkovski. *Interoperable and Distributed Processing in GIS*. Taylor & Francis, 1998.

A. Včkovski and F. Bucher. Virtual Data Sets—Smart data for environmental applications. In *Proceedings 3rd International Conference/Workshop on Integrating GIS and Environmental Modeling*, Santa Fe, USA, 1996.

D. F. Watson. *Contouring: A guide to the analysis and display of spatial data*. Pergamon Press, Oxford, UK, 1992.

A. Woolf and D. Lowe. Climate science modelling language version 2—user's manual. `http://ndg.badc.rl.ac.uk/csml/`, 2007.

W. Zhang and G. J. Hunter. Temporal interpolation of spatially dynamic objects. *GeoInformatica*, 4(4):403–418, 2000.

A Examples of FieldGML files

A.1 Two-dimensional Points (ScatteredPoints)

This is an excerpt from a FieldGML file containing 5000 scattered points, representing the elevation of a mountain. The interpolation method is natural neighbour method, which does not require any user-defined parameters.

```
<?xml version="1.0" encoding="utf-8"?>
<Field xmlns="http://www.gdmc.nl/ledoux/fieldgml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.gdmc.nl/ledoux/fieldgml
http://www.gdmc.nl/ledoux/fieldgml/fieldgml.xsd">
  <Samples dimension="2">
    <Attribute>
      <Name>elevation</Name>
      <Scale>Continuous</Scale>
    </Attribute>
    <ScatteredPoints>
      <PointValue>
        <gml:pos>199.129206765 108.59756162</gml:pos>
        <value>87.7276930119</value>
        <gml:TimeInstant>
          <gml:timePosition>2003-02-13</gml:timePosition>
        </gml:TimeInstant>
      </PointValue>
      <PointValue>
        <gml:pos>166.318409087 183.304411542</gml:pos>
        <value>70.1089874139</value>
      </PointValue>
      <PointValue>
        <gml:pos>190.09899811 248.528765249</gml:pos>
        <value>43.2618775452</value>
      </PointValue>
      .
      .
      .
  </Samples>
  <InterpolationMethod>
    <NaturalNeighbour/>
  </InterpolationMethod>
```

```
</Field>
```

A.2 Two-dimensional Points (TessPointsRule)

This is an excerpt from a FieldGML file containing 5000 scattered points, but they are stored in a TessPointsRule of type "Delaunay", which means that they have to be tessellated. The interpolation method attached is piecewise linear. This dataset is thus a standard TIN.

```
<?xml version="1.0" encoding="utf-8"?>
<Field xmlns="http://www.gdmc.nl/ledoux/fieldgml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.gdmc.nl/ledoux/fieldgml
http://www.gdmc.nl/ledoux/fieldgml/fieldgml.xsd">
  <Samples dimension="2">
    <Attribute>
      <Name>elevation</Name>
      <Scale>Continuous</Scale>
    </Attribute>
    <Tessellation>
      <TessPointsRule>
        <ScatteredPoints>
          <PointValue>
            <gml:pos>199.129206765 108.59756162</gml:pos>
            <value>87.7276930119</value>
          </PointValue>
          <PointValue>
            <gml:pos>166.318409087 183.304411542</gml:pos>
            <value>70.1089874139</value>
          </PointValue>
          <PointValue>
            <gml:pos>190.09899811 248.528765249</gml:pos>
            <value>43.2618775452</value>
          </PointValue>
          .
          .
          .
          <PointValue>
            <gml:pos>189.548967103 72.1125642931</gml:pos>
            <value>123.311568705</value>
          </PointValue>
        </ScatteredPoints>
        <TessRule>Delaunay</TessRule>
      </TessPointsRule>
    </Tessellation>
  </Samples>
```

```
  <InterpolationMethod>
    <Piecewise>Linear</Piecewise>
  </InterpolationMethod>
</Field>
```

A.3 Three-dimensional Points

This is an excerpt from a FieldGML file containing 150 scattered points in 3D, representing a geological body.

```
<?xml version="1.0" encoding="utf-8"?>
<Field xmlns="http://www.gdmc.nl/ledoux/fieldgml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.gdmc.nl/ledoux/fieldgml
http://www.gdmc.nl/ledoux/fieldgml/fieldgml.xsd">
<Samples dimension="3">
    <Attribute>
      <Name>concentration of a xxx chemical</Name>
      <Scale>Continuous</Scale>
    </Attribute>
    <gml:TimeInstant>
      <gml:timePosition>2003-02-13</gml:timePosition>
    </gml:TimeInstant>
    <ScatteredPoints>
      <PointValue>
        <gml:pos>0.13385 0.35097 0.50739</gml:pos>
        <value>0.50739</value>
      </PointValue>
      <PointValue>
        <gml:pos>0.13385 0.35097 0.61167</gml:pos>
        <value>0.61167</value>
      </PointValue>
      <PointValue>
        <gml:pos>0.13385 0.35097 0.73074</gml:pos>
        <value>0.73074</value>
      </PointValue>
      .
      .
      .
      <PointValue>
        <gml:pos>0.13385 0.35097 0.83969</gml:pos>
        <value>0.83969</value>
      </PointValue>
  </Samples>
  <InterpolationMethod>
    <NearestNeighbour/>
  </InterpolationMethod>
```

```
</Field>
```

A.4 Contours Lines

This is an excerpt from a FieldGML file containing 5000 scattered points, representing the elevation of a mountain.

```
<?xml version="1.0" encoding="utf-8"?>
<Field xmlns="http://www.gdmc.nl/ledoux/fieldgml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.gdmc.nl/ledoux/fieldgml
file:/home/hugo/projects/standards/fieldgml/fieldgml.xsd">
  <Samples dimension="2">
    <Attribute>
      <Name>elevation</Name>
      <Scale>Continuous</Scale>
    </Attribute>
    <ScatteredLines>
      <LineSegmentValue>
        <gml:posList>110.990258621 119.169396552
          108.26387931 119.169396552 104.174310345
          117.654741379 102.508189655 115.231293103
          101.599396552 113.110775862 102.356724138
          110.990258621 102.508189655 108.566810345
          103.568448276 106.749224138 105.234568966
          104.477241379 106.900689655 102.659655172
          110.081465517 99.9332758621 112.959310345
          ...
        </gml:posList>
        <value>150.0</value>
      </LineSegmentValue>
      <LineSegmentValue>
        <gml:posList>94.0261206897 153.930732759 90.3909482759 ...
        </gml:posList>
        <value>140.0</value>
      </LineSegmentValue>
      .
      .
      .
      <LineSegmentValue>
        <gml:posList>42.5570933413 2.0839402497 46.6017449465
          4.84165725327 50.094853151 5.02550505351 56.3456783591
          5.02550505351 61.8611123662 4.10626605232 67.0088507729
          2.45163585018
        </gml:posList>
        <value>70.0</value>
      </LineSegmentValue>
```

```
    </ScatteredLines>
  </Samples>
  <InterpolationMethod>
    <IDW>
      <Power>2</Power>
        <SearchParam>
          <Radius>12</Radius>
</SearchParam>
    </IDW>
  </InterpolationMethod>
</Field>
```

A.5 Raster File in 2D

This is an excerpt from a FieldGML file representing a raster file (GeoTiff format). The interpolation is piecewise constant, that is when interpolating the value of the pixel is used.

```
<?xml version="1.0" encoding="utf-8"?>
<Field xmlns="http://www.gdmc.nl/ledoux/fieldgml"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml="http://www.opengis.net/gml"
xsi:schemaLocation="http://www.gdmc.nl/ledoux/fieldgml
http://www.gdmc.nl/ledoux/fieldgml/fieldgml.xsd">
    <Samples dimension="2">
        <Attribute>
            <Name>elevation</Name>
            <Scale>Continuous</Scale>
        </Attribute>
        <Array>
            <GridFile>
                <gml:Grid dimension="2">
                    <gml:limits>
                        <gml:GridEnvelope>
                            <gml:low>0 0</gml:low>
                            <gml:high>125 125</gml:high>
                        </gml:GridEnvelope>
                    </gml:limits>
                    <gml:axisName>x y</gml:axisName>
                </gml:Grid>
                <fileName>/home/hugo/data/fieldgml/5.tif</fileName>
            </GridFile>
        </Array>
    </Samples>
    <InterpolationMethod>
        <Piecewise>Constant<Piecewise/>
    </InterpolationMethod>
</Field>
```