FOSS4G DENVER 2011

12-16 SEPTEMBER 2011 SHERATON DENVER DOWNTOWN
2011.FOSS4G.ORG

Conference Proceedings

Free and Open Source Software for Geospatial Conference

OSGeo Journal
Vol. 10 - Feb 2012

Volume 10 Contents

FOSS4G 2011 Conference Proceedings **2**
Editorial - FOSS4G 2011 Academic Chair 3
Open source based online map sharing to support real-time collaboration 5
An innovative Web Processing Services based GIS architecture 15
Using GRASS and R for Landscape Regionalization through PAM Cluster Analysis 26
Functional Coverages 32
Opticks Open Source Remote Sensing and Image Processing Software 45
Implementation, challenges and future directions of integrating services from the GIS and decision science domains 50
A Vivid Relic Under Rapid Transformation . . . 56

Welcome from the Conference Chair

Welcome to this special edition of the OSGeo Journal, featuring selected papers from the academic track that were presented at the FOSS4G (Free and Open Source Software for Geospatial) 2011 conference in Denver.[1] The conference was the largest FOSS4G yet, with 914 attendees from 42 countries. Feedback from attendees was very positive, with the post-conference survey giving it an overall rating of 4.32 out 5. The attendance reflects the strong growth in interest in open source software that we are currently seeing in the geospatial industry.

We made a conscious effort in 2011 to enhance the academic track at the conference by providing improved publishing opportunities. We did this through publishing papers both in "Transactions in GIS" and in this edition of the OSGeo Journal. I would like to thank Rafael Moreno for leading this effort, as well as the rest of the organizers of the academic track who Rafael recognizes below.

Peter Batty, Ubisense
FOSS4G 2011 Conference Chair

[1] FOSS4G: http://foss4g.org

FOSS4G 2011 Conference Proceedings

Functional Coverages

Gennadii Donchyts, Fedor Baart, H.R.A (Bert) Jagers, Arthur van Dam

Abstract

A new Application Programming Interface (API) is presented which simplifies working with geospatial coverages as well as many other data structures of a multi-dimensional nature. The main idea extends the Common Data Model (CDM) developed at the University Corporation for Atmospheric Research (UCAR). The proposed function object model uses the mathematical definition of a vector-valued function. A geospatial coverage will be expressed as a vector-valued function whose dependent variables (the vector components) are fully defined by its independent variables (the coordinates).

Our goal is to provide an API using a terminology and an object model that is both appealing to computer scientists and numerical modelers and is flexible enough to enable defining data structures for a wide range of applications. Examples of such data structures can be: wind velocity as a continuous variable defined along the channels in a river network. Precipitation data defined as a time-dependent variable on a set of sub-catchments of a drainage basin, preserving association with sub-catchment features.

The new object model provides a basis for both continuous and discrete coverages including non-geospatial data structures such as time series. Different storage models for variables are implemented, based on the Network Common Data Format (NetCDF), the Geospatial Data Abstraction Library (GDAL) and memory.

The API is available as set of open source libraries developed in C# consisting of a multi-dimensional arrays library; a scientific data structures library defining variables, functions, units of measure; a geospatial extensions library built on top of GeoAPI.NET and NetTopologySuite, defining specialized coverages: network coverage, feature coverage, regular grid coverage, and unstructured grid coverage.

1 Introduction

Geospatial coverage is the concept describing geographic phenomena upon which a range of data values can be present. The concept dates back to early ArcInfo[35] versions, but has been applied and rethought numerous times in data standards and Application Programming Interface (API)s ever since. The assumptions that were made in these do not alway make them easily applicable in new fields, where both the data and the spatial domains may be more complicated. One special example is the role of the time dimension, which was absent in early coverage definitions. Over the past decades, many initiatives were taken to incorporate time into geospatial applications ((Peuquet, 1999), (Wachowicz, 1999), (Goodall et al., 2004)) resulting in a sometimes overly special role of the time dimension.

The goal of the present work is to define a high-level API that will allow presenting most of the existing geospatial and temporal coverage data types in terms of vector-valued functions of one or more independent variable(s), inspired by basic vector calculus. The main reason for doing this is a unification of the storage of spatiotemporal domains and coverage values on top of these domains. In addition this will simplify development of generic data transformation algorithms such as aggregations, filtering, interpolation, and extrapolation. It will be much easier to re-use them, which will simplify development of applications, as was already shown for example for the Open Modeling Interface (OpenMI) 2.0 by Donchyts et al. (2010). We will show that this higher-level API can not only define scalar data on spatial networks or grids, but can also be used to define more general data structures, such as time-series or vector fields. Evidently, these can be used in geospatial applications, but are also applicable in a wider scope.

Related work

The definition and use of a geospatial coverage can be approached from several angles; two existing standards are the ISO 19123 standard (schema for coverage geometry and functions) and the Common Data Model (CDM), designed by the University Corporation for Atmospheric Research (UCAR).

The ISO 19123 standard defines a coverage as a "coverage is a feature that associates positions within a bounded space (its domain) to feature attribute values (its range)". ISO 19123 as well as the Open Geospatial Consortium (OGC) Abstract Specifications (2007) define an abstract concept of the coverage object model and how it supports mapping from a spatial, temporal or spatio-temporal domain to feature attribute values. These standards are mainly used to form the basis of an geospatial API's such as GeoAPI.[36]

Alternatively, the CDM API mainly focuses on the multi-dimensional aspects of coverages. An overview of the UCAR CDM and its mapping into the corresponding elements of the international standard coverage data model of ISO 19123 is presented by Nativi et al. (2008). More information about CDM and the Network Common Data Format (NetCDF) can be found in Rew and Davis (1990).

As a result of these two different angles, the coverage API is very well suited in geospatial applications, whereas the CDM API has proven itself in numerical applications and for large datasets, see, e.g., Rutledge et al. (2006) and Signell et al. (2008), respectively. For applications where data structures are less organized (ecological models) or require more complex relations (river models), both data models are less applicable. This paper deals with datasets where information is dependent on time and covers a network- or grid-based area (the spatial domain). Examples are: ocean dynamics (water levels and velocities in 3D grid layers) and river flows (water levels, velocities and transport across network connections), see, e.g., Kernkamp et al. (2011). The new API should facilitate this.

An example of an often-used API for *time-dependent* data sets defined on complex geometries can be found in the open source Visualization Toolkit (VTK) (Schroeder et al., 2000).

[35] http://www.esri.com/software/arcgis/arcinfo
[36] http://www.geoapi.org/

Each object in VTK maintains an internal time stamp that is automatically updated when the object state changes (usually as a result of setting an instance variable value). An alternative approach to dealing with the time dependent grid based data is presented by Howe and Maier (2005). They use a relational database based approach using an algebra notation to manipulate both regular and irregular gridded datasets. When comparing these API's and data models we can see that they differ in scope, detail and field. When approached from the visualization field as was done in VTK, the time dependency and geographic aspects were added later (see for example vtkGeoGlobeSource, vtkTemporalDataSet). Because the API is focused on visualizations we can see that all geometry objects are well defined and easy to comprehend.

When approached from the Geographic Information System (GIS) field we can see that the Feature is the base of all relevant objects, however it is still ongoing discussion about what it is and how it should be implemented and used in applications. OGC Coverage Primer (Nordgren, 2006) reflects this in the following way: *The question "What is a feature?" leads directly to a philosophical rabbit hole which deposits the unwary questioner in a wonderland from which it is difficult to return*. Also, time aspects are still being introduced in the OGC standards and as a result most widely used geospatial API's that build upon these OGC Standards still lack support for time dependency.

It is important to note that computer scientists will often define their own API and naming conventions that are (partly) based on the problem domain in which they happen to be working at the time rather than using the terminology of the generic underlying concepts. This complicates the reuse by people in other domains. The idea to use multi-valued functions to represent data structures in a more generic way has been formulated by Treinish (1999) as: "Any data set may be considered as a single or multi-valued function of one or more independent variable(s)". In the present work we try to generalize and expand this idea to be applicable to geospatial coverages, resulting in both a conceptual description and an implementation in the form of a class library which can be easily re-used in geospatial applications.

Outline

Section 2 motivates the approach we took in defining our API and summarized the underlying ideas. Sections 3 and 4 then respectively describe the generic vector-Function API and the geospatial API on top of it. Section 5 describes how the functions and their values can be stored in memory and in files. Section 6 considers our resulting API and provides some additional motivation. Section 7 summarizes the paper in several conclusions. The appendices summarize the terminology and acronyms used throughout the paper.

2 Method

The most important question is how to define an API that offers data types and functionality for generic mathematical concepts such as variables, and at the same time be specifically suitable as well for both geospatial and non-geospatial applications? Can we introduce time-dependency in a non-intrusive way? The idea is to define a common API which will describe all generic mathematical data structures required to manage variable values and and then base a geospatial coverage API on top of that to provide a better separation of concepts and better code re-use.

The spatial domains also pose challenges: what to do when the topology is complicated, for example a river network? The channels in this network form a set of interconnected features, each of which is a polyline or polygon geometry as shown in Figure 1. Can we still use the same classes to model this situation in a similar way as we would do it for a simple time series at a single point station?

Next comes the data defined on the domains. In our river example, consider a time-dependent wind velocity field defined as a continuous vector variable $\vec{V} = (v_x, v_y)$ (discrete values with interpolation) along the channels in the river network. Inter- and extrapolation on such complicated geometries is also nontrivial. Do we need to deal with spatial variables differently than with time variables?

Even if it would be clear what should be done in this example, it is still not a trivial task to define using an object-oriented language. Both the OGC Coverage model and CDM fail to define it completely. The OGC Coverage model is not flexible enough in the sense that it introduces many classes but there is no conceptual basis where every type of Coverage would fit. In terms of the CDM, it should be trivial to define values of all variables used here (river coordinates and wind) using multi-dimensional arrays. However, there is no room for the rest of the meta-information (the river topology and offsets of network locations). The use of basic CDM attributes is insufficient here.

We will show that using the new approach proposed in this paper all coverage types can be defined in terms of vector-valued functions.

Design steps

Before introducing an API for multi-dimensional data structures, we will first try to identify and analyze the actual problem domains related to the practical applications that involve coverages or multi-dimensional data structures (Figure 2). Then we will try to identify the functionality required by the developers when developing applications related to these domains.

For the design of the API we used the following steps: identification of entities / classes, construction of a reference implementation of the classes to match the different fields (geospatial, environmental), separation of core logic of the classes from persistency so that multiple storage choices can be used, identification of interfaces / classes which should belong to the API.

The term Coverage is used in the geospatial domain (by the OGC) to describe discrete or continuous characteristics of the real world features. We will try to match our API as much as possible (on a conceptual level) to the requirements listed in the OGC standards.

Before moving to the geospatial domain we will first try to analyze in details how vector-valued function can be defined in terms of the software component since we plan to use it as a basis of our API. After that we will show that the new API is very well suitable to describe any coverage used in the geospatial domain.

When developing an API for multi-dimensional structure an important aspect is persistency. In section 5 Persistence

How to define an API so that it will allow most closely to define this coverage function (time-dependent vector field defined at selected locations along the network) ?

$\vec{V}(t, nl) = (v_x, v_y)(t, nl)$
$nl = (c, l)$

nl – network location
c – curve
l – distance along the curve

How to interpolate values between points?

How to interpolate values between times?

How to interpolate values at junctions?

$v_x(t)$

How to query time-series at a specific location in the same way for coverage defined on a network or on a regular grid? Preserving a link to the source coverage?

Figure 1: Problems appearing while modeling variables defined on the network

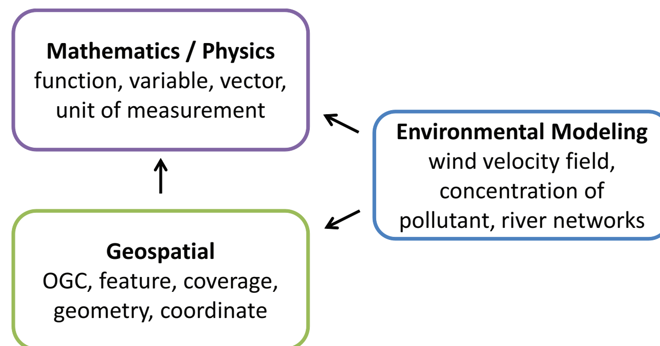


Figure 2: Problem domains involved

several storage options will be discussed allowing reuse of existing data access libraries or file formats to persist structure and values of the vector-valued functions.

We will use motivating example problems appearing in the environmental modeling field, see sections 3.2 and 4.

3 Vector-valued Function

3.1 Mathematical Definition

The main idea of the new data structure is based on the following statement:

Any data structure of a multi-dimensional nature can be presented as a single or multiple number of a vector-valued

functions.

In vector calculus a vector-valued function is defined in general form as Equation 1

$$F = (f_1, f_2, \cdots, f_n)(x_1, x_2, \cdots, x_m) \quad (1)$$

where $x_1, x_2, \cdots, x_m$ are independent variables and $f_1, f_2, \cdots, f_n$ are dependent variables.

Typical examples of independent variables are time, spatial coordinates, categories, etc. Dependent variables usually represent actual quantities. In scientific applications these variables are very frequently sampled or discretized and thus their values are defined in a form of arrays together with information about how to interpolate values of dependent variables where no exact values of the independent variables are available. For arguments (independent variables) these arrays are in most cases one-dimensional ($R1$) while for components (dependent variables) dimensionality of arrays is defined by a number of arguments used (Rm). In many cases value type used by the variable is a real number (R) however in general it can be any type available in programming language, for example: string, class, feature, etc. In some cases the rank of the independent variable space may be different from the rank of the sampled independent variable space: discrete coordinate arrays may be rank 1 (regular grid), or rank 2 (curvilinear/irregular grid), or even rank 3 (time dependent moving grid).

Let us analyze vector-valued function in more details: consider that we want to define a time-dependent velocity field (e.g. 3 moments in time), defined on a discrete, regular grid (see Figure 3). In order to store values of this function we will have first to decompose it into independent (x, y and t) and dependent (vx and vy) variables. We can see that independent variables have to be defined as a set of values (ordered set if we want to interpolate values of the dependent variables along the argument). On the other hand dependent variables require 3-dimensional arrays to store their values (number of independent variables used in function).

It is simple to show that every variable can be also considered as a function, as result we can list all objects required to store the above example, as can be seen in Table 1.

In the table Arguments denote independent variables and components - dependent variables.

In general any variable used in a function (independent or dependent variable) can be described by a set of properties: value type, units of measure, typical minimum and maximum values.

Additionally, for every independent variable we need to define interpolation and extrapolation method since its values are defined only at discrete locations. This will allow computing values of dependent variables outside of independent variable values space.

From the table we can see that in order to completely define all objects used in this example we will have to define all functions listed in the table, together with their properties as well as relations between them such as that some of them are used as independent variables and other as dependent variables. In case if we have a function that uses more than one component (V) - the only thing to be stored is its relation to child component variables since its values are completely defined by the values of its components (v_x, v_y)

3.2 API

Based on the points discussed in the previous section we believe that the class diagram presented in Figure 4 most closely describes all objects required to introduce a vector-valued function.

As can be seen from Equation 1, component variables can be actually seen as vector-valued functions by themselves.

The code listing in Figure 5 shows how the API can be used. This is a simple example that shows how we can define a variable, its properties and an array of values.

Since a Variable in our API is automatically considered to be a Function - we can also start combining variables as shown in Figure 6.

This type of function is one of the most frequently used. A simple example from hydrodynamic modeling can be a water level defined as a function of time, e.g. measured at some location: y=y(t). In this case value type of the argument variable will be DateTime (C#) instead of double. In this case a water level is dependent variable and time is independent variable.

If we make it a bit more complex we can measure water level on a moving boat. In this case a water level variable is measured as a function of time, but so is location. So we have location as an object and a water level as a scalar value defined as a function of time: *F=(location, depth)(t)*.

Suppose we measure wind direction at a meteorological station, then we have two parameters for direction and speed or in Cartesian space a u and v part of the vector. Both again are defined as functions of time.

The source code required to work with a vector-function that uses more than one independent variable (components) does not look much different, see Figure 7.

Note that API provides different ways to access or assign values of the variables. Depending on performance requirements values of the variable can be set as an array at once or one by one using simple and intuitive syntax.

4 Adding geospatial aspects to Function

Even though the above API is powerful enough to describe a vector-valued function, in some cases we need to extend it in order to apply it to other domains. We will try to define a Coverage types on top of the Function API.

4.1 What is Coverage?

Definition of Coverage in many geospatial applications is also confusing and is given on a very conceptual level. We will introduce term Coverage as a bridge between two worlds: Geospatial and Mathematical. The UML class diagram is shown in Figure 8. The nice thing is that if Coverage can be defined as a function - then its values can be accessed in the same way as in examples of the previous section. On the other side Coverage is a geospatial object, which means it has to extend functional part with geometries. For example Geometry property of the coverage can be either a geometry that defines bounds of the coverage, or a complex geometry representing every location where values of the coverage are defined (GeometryCollection).

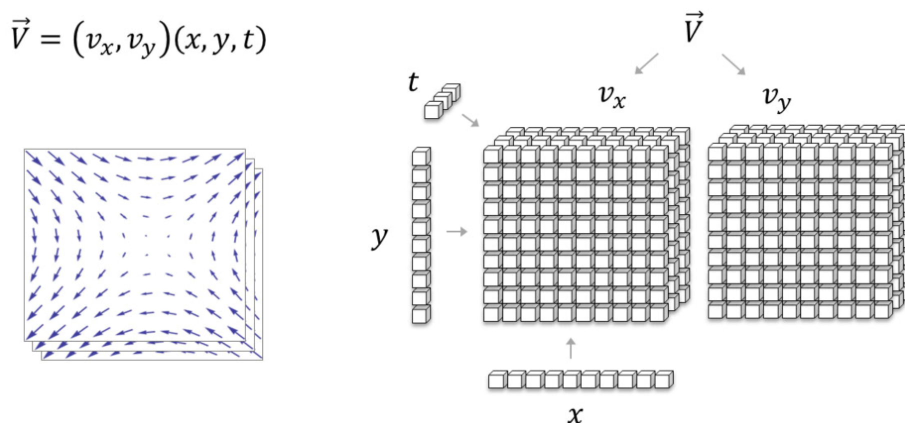


Figure 3: Analysis of the data structures required to store discrete vector-valued function

Table 1: List of functions and their properties used in example

Function	Definition	Arguments	Components	Rank	Unit
x	(x)	0	1	1	L
y	(y)	0	1	1	L
t	(t)	0	1	1	T
v_x	$(v_x)(x, y, t)$	3	1	3	L/T
v_y	$(v_y)(x, y, t)$	3	1	3	L/T
$\vec{V}$	$(v_x, v_y)(x, y, t)$	3	2	3	L/T

In the OGC standards there is a clear distinction made between continuous and discrete coverages. We do not see the need to separate them. Really, the difference between if coverage is discrete or continuous is just an interpolation type used for its arguments. In fact we can define a Coverage that is discrete along one argument and continuous (interpolated) along the other one.

All other coverage types can be very simply defined as an extension of the ICoverage interface. For the sake of space we will not list all UML diagrams here.

4.2 Regular Grid Coverage

In many applications, including numerical modeling, data of the models are often defined on a discrete grid. Suppose we compute water level which can change in time and is defined on a rectangular regular grid, then it can be defined as a function of x, y coordinates and time t. In these case x and y identify location and used as independent variables as showin in Figure 9. Actually this is also true for rectilinear grid, where values of x and/or y are not equidistant, see (Balaji and Liang, 2006). In case of regular grid the values of x and y variables (arguments) are equidistant and as a result their storage can be simplified.

Depending on a type of grid we can also use cells of the grid (objects) as an independent variable values instead of scalar x and y variables to identify location on a grid. See 4.5 section for an example.

The biggest advantage of using the same base API to work with Coverages is that the functionality of Coverage can be very easily extended. For example in order to make regular grid coverage time dependent we only need to add an additional argument (independent variable) of a time value type, see Figure 10. The rest remains the same.

Of course in case if we have other functionality (for example rendering) based on a specific Coverage type - we will need to extend it a little to make sure that we accessing only with a values corresponding to a single time value.

4.3 Feature Coverage

We will call FeatureCoverage a function where one of arguments uses a Feature as a value type.

Consider the following class as an example: a City, which has some default properties such as Name and Population (these properties are also available as feature attributes and accessible via Attributes dictionary).

Now imagine that we want to compute a total precipitation over city for a given period of time without modifying existing City features. It can't be simply added as an attribute (a property) since it doesn't seem to be a default characteristic of the City. In this case FeatureCoverage type can be used to define a coverage function that uses cities as values of an independent variable as show in Figure 11 and Figure 12.

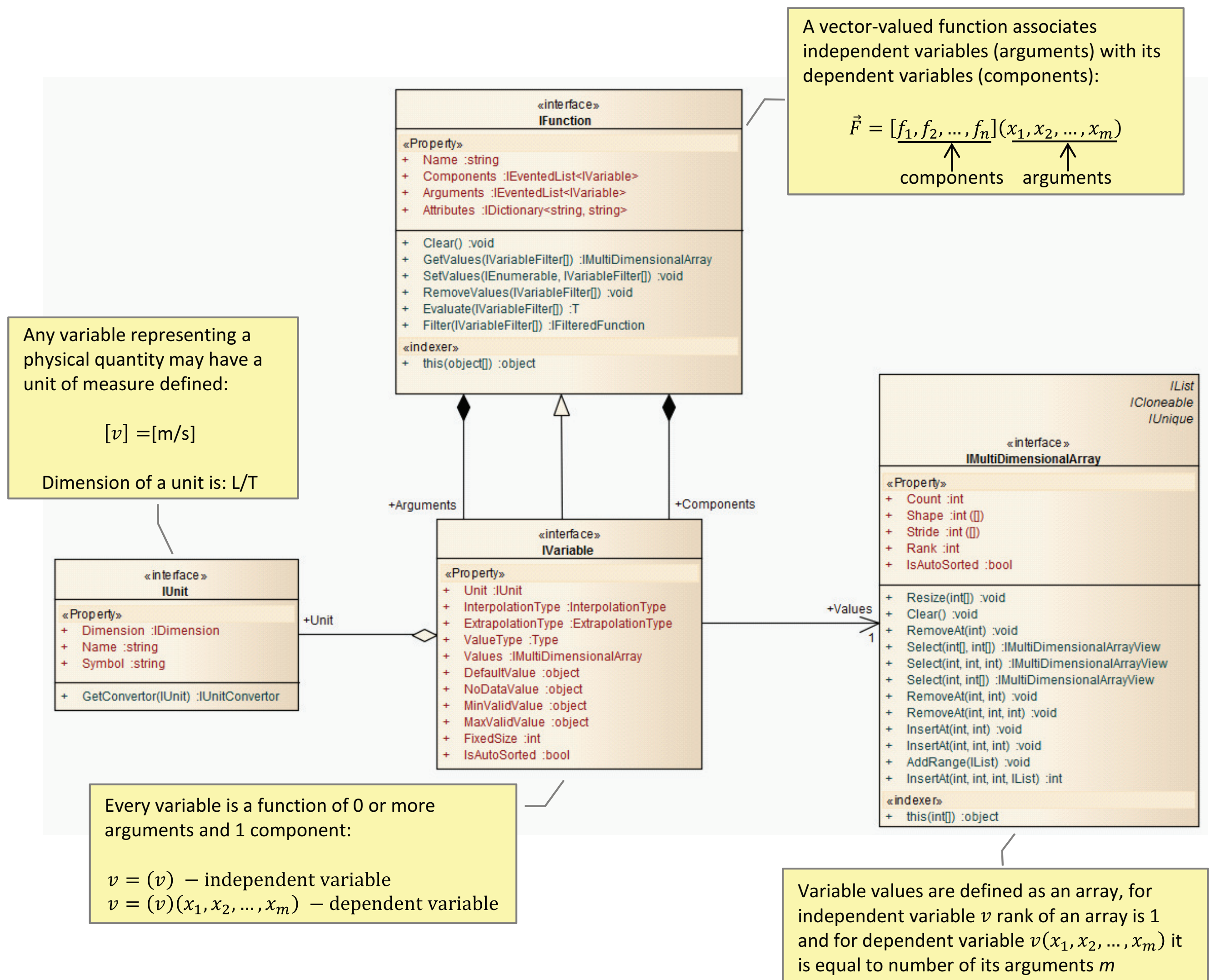


Figure 4: Vector-value function class diagram

4.4 Network Coverage

In some cases variable values are defined on a set of curves (see OGC CurveCoverage). In more specific cases these curves are connected into a network or graph, for example: river networks, roads and pipelines. It is frequently necessary to define a Coverage that can be seen as a continuous function defined along the branches (the polylines) of such a network (see Figure 13).

This example is a bit more complicated compared to the ones discussed in the previous sections. The main problem is that we do not have an explicit independent variable to depend on in our function. Still we can introduce a variable which is defined as a combination of a curve and offset along that curve. In this case it will uniquely define location on a network.

The only remaining problem is that if we would like to evaluate values of the dependent variables in other but existing locations on a network - we will need some custom interpolation algorithm at the sections where branches connect with each other. Once it is implemented - the rest of the functionality works as in any other function type.

Additionally to the network location independent variable we can also add time or any other variable.

We've defined NetworkCoverage as a separate coverage type as shown in Figure 14. However it will be more correct to introduce a CurveCoverage type first, and then define NetworkCoverage as an extension to it, in case if we need to use some network-specific interpolations e.g. at the nodes connecting different branches of the network. In this case it will be more consistent with the ideas introduced in the OGC Coverage standards.

4.5 Unstructured Grid Coverage

We did not fully implement support for unstructured grid coverages yet using new API, but it should not be any more complicated than previously shown examples. In fact UnstructuredGridCoverage can be implemented in a way similar to FeatureCoverage, which depends on grid cells or interfaces between cells, depending on where values are defined (see Figure 15). Additionally, custom interpolation methods have to be implemented for e.g. IDW, Krigging or any other interpolation methods required to evaluate values outside of the

```
[Test]
[Category("Example")]
public void SimpleVariable()
{
    var x = new Variable<double>
    {
        Name = "x",
        Unit = new Unit("meter", "m"),
        Values = { 1.0, 2.0, 3.0 }
    };

    // access values array used by variable x
    IMultiDimensionalArray<double> values = x.Values;

    // asserts
    values[1]
        .Should("2nd value").Be.EqualTo(2.0);

    values.Rank
        .Should("rank of a values array is 1").Be.EqualTo(1);

    values.Count
        .Should("number of values").Be.EqualTo(3);
}
```

$x = \{1.0, 2.0, 3.0\}$

unit of measure of *x* is *meter*

Figure 5: Create and set values of a simple scalar variable (C#)

```
[Test]
[Category("Example")]
public void DependentVariable()
{
    var x = new Variable<double> { InterpolationType = InterpolationType.Linear };
    var y = new Variable<double> { Arguments = { x } };

    // set values using value index operator
    y[1.0] = 100.0;
    y[2.0] = 200.0;
    y[3.0] = 100.0;

    // set values using method (replaces existing values of y)
    y.SetValues(new [] { 100.0, 200.0, 300.0});

    // access values using value index operator
    y[1.0]
        .Should("exact value y(x = 1.0) = 100.0").Be.EqualTo(100.0);

    y[1.5]
        .Should("interpolated value y(x = 1.5) = 150.0").Be.EqualTo(150.0);

    // access values of y as an array
    x.Values
        .Should("check x values").Have.SameSequenceAs(new[] { 1.0, 2.0, 3.0 });

    y.Values
        .Should("check y values").Have.SameSequenceAs(new[] { 100.0, 200.0, 300.0 });
}
```

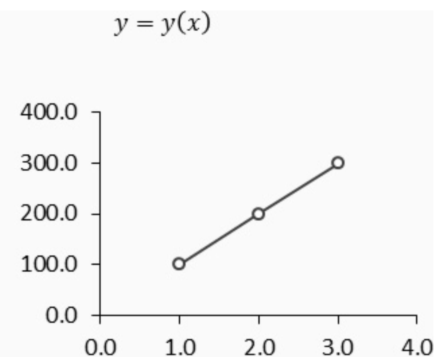


Figure 6: Create a 1d function y=y(x), use linear interpolation for x (C#)

argument values domain.

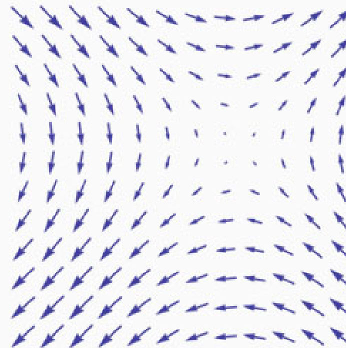
5 Persistence

In order to store values of the functions an interface IFunctionStore was introduced as a part of API, see Figure 16. An implementation of the IFunction / IVariable uses IFunctionStore to access all functions available in the store as well as to access their values.

Currently the following implementations are supported: MemoryFunctionStore, NetCDFFunctionStore and GdalFunctionStore. The first implementation is a default one and simply keeps a set of multi-dimensional arrays as well as a set of function objects in memory. The second is used to store functions in the NetCDF files, wrapping UCAR Java implementation converted to .NET on a byte-code level using IKVM.NET.

```
[Test]
[Category("Example")]
public void VelocityField2D()
{
    var x = new Variable<double> { Values = { 1.0, 2.0 } };
    var y = new Variable<double> { Values = { 1.0, 2.0, 3.0 } };
    var vx = new Variable<double>();
    var vy = new Variable<double>();

    var velocity = new Function
                       {
                           Components = { vx,  vy },
                           Arguments = { x, y }
                       };

    // set vx, vy values as an array
    vx.SetValues(new[] { 100.0, 200.0, 300.0, 400.0, 500.0, 600.0 });
    vy.SetValues(new[] { 100.0, 200.0, 300.0, 400.0, 500.0, 600.0 });

    // set vx values using indexer
    vx[1.0, 1.0] = 100.0; vx[2.0, 1.0] = 400.0;
    vx[1.0, 2.0] = 200.0; vx[2.0, 2.0] = 500.0;
    vx[1.0, 3.0] = 300.0; vx[2.0, 3.0] = 600.0;
}
```

$\vec{V} = (v_x, v_y)(x, y)$



	Arrays
v_x	[][]
v_y	[][]
x	[]
y	[]

Figure 7: Define 2d velocity field function (C#)

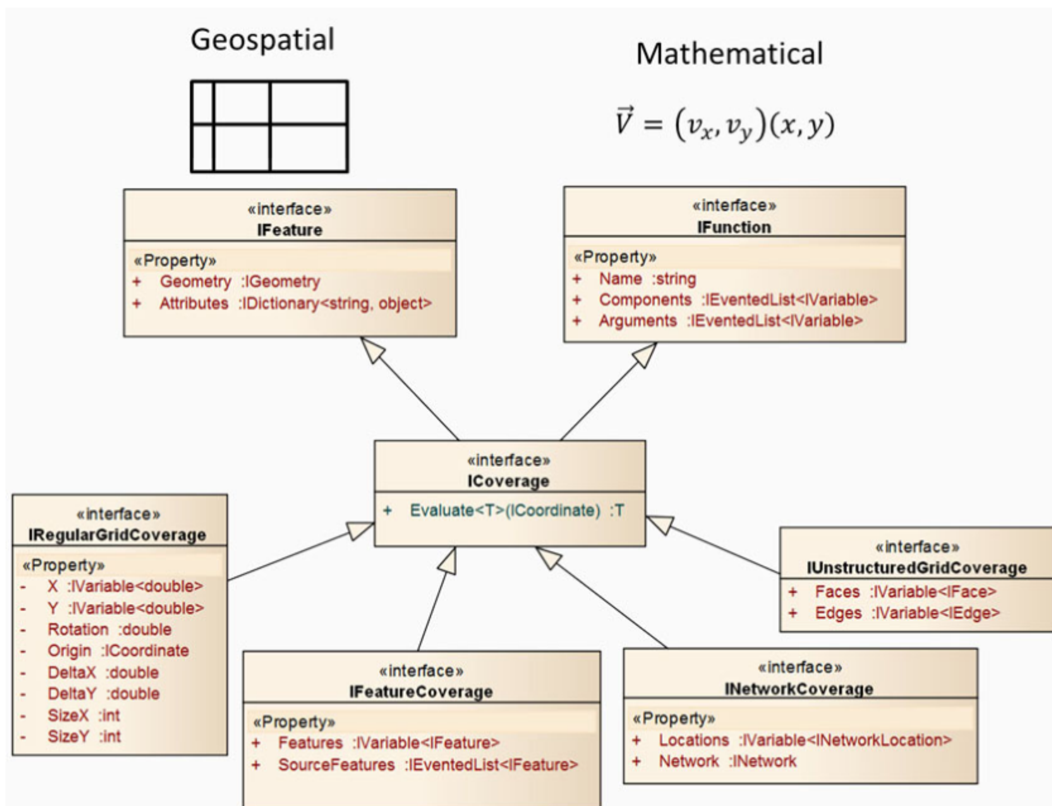


Figure 8: Coverage Class Diagram

The third implementation is used to access raster data stored in GDAL file formats.

5.1 Store Functions in NetCDF

Because the Common Data Model is a concept behind the NetCDF file format, the file format is almost a perfect match for as a storage for Function classes. Still the following information needs to be defined implicitly using attributes:

Relations between container vector-valued function and its component variables.

Relations between components and arguments. Even it can be reconstructed using NetCDF dimensions; it is still error prone and not very intuitive.

Custom type mappings, in case if we want store entities (objects) in the NetCDF variables.

In the last case NetCDF attributes can be used using some

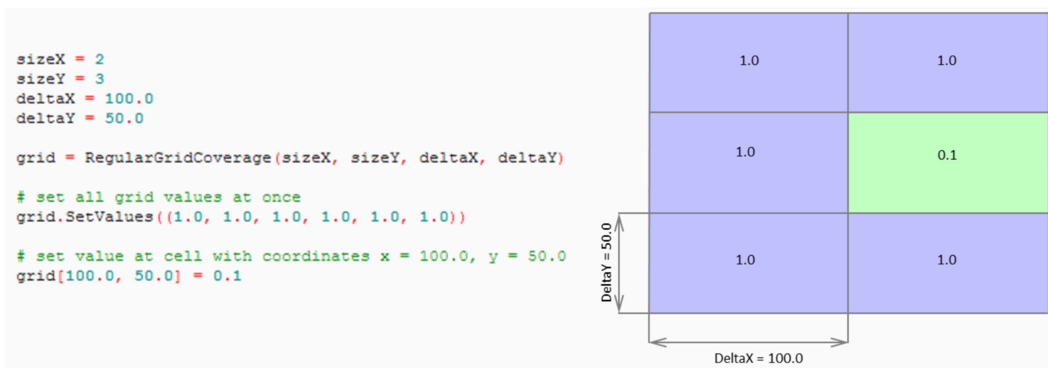


Figure 9: RegularGridCoverage defined as 2D function (Python)

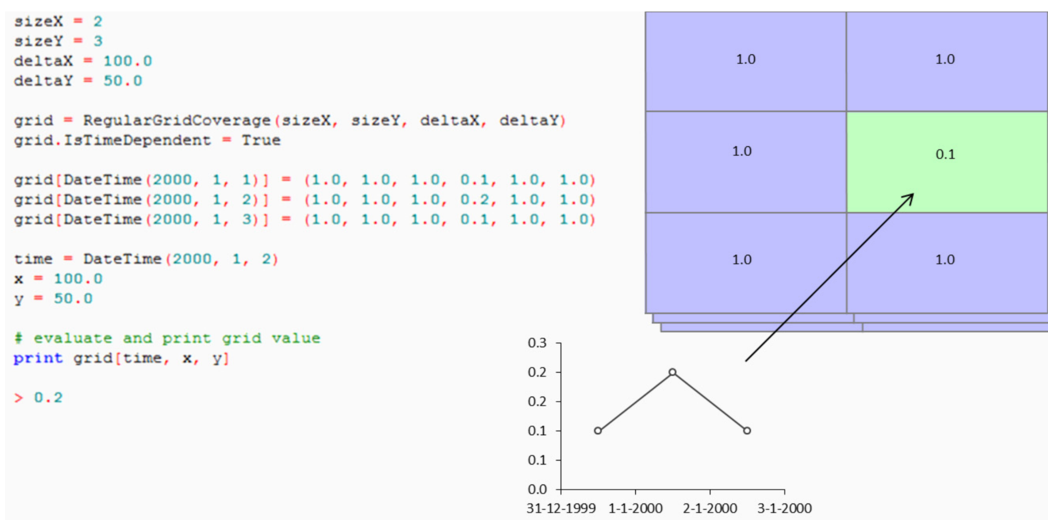


Figure 10: Define and query value of a time-dependent RegularGridCoverage (Python)

convention in order to define where properties need to be stored and to which type they belong to.

6 Discussion

The need to this API was mainly dictated by the reality that there was no API library available written in C# which is comparable to NetCDF. Instead of reinventing the wheel we tried to extend concepts used in the CDM. On the other hand we did not port (or wrap) existing NetCDF API in C# but tried to make it more intuitive by reviewing the concepts behind it. The NetCDF library is a very powerful one but we believe that the API introduced here better represents the reality (or mathematical abstractions used to describe the reality within computerized applications). The major difference between CDM and the present API, except of course the language and syntax, is that CDM uses Dimensions as a separate entity next to Variable. For the API we present the Dimension is not required as a separate entitity. The dimensions can be derived from the vector-valued functions.

Some variables need to be defined not as a set of values but e.g. as an equidistant series with start, stop and step. In this case only several properties need to be stored instead of array. The implementation can still generate all possible values on-the-fly in order to use this specific variable type in the same way as other variables.

In some cases a single vector-valued function is not sufficient to define all data structures. For example for curvilinear grid we want to preserve the information that the grid cells are defined as a 2D matrix. In this case it will be necessary to combine different vector-valued functions in order to fully define the data structure.

When comparing the proposed API to the OGC/ISO Coverage specifications it is important to note that the latter specifications are quite complex compared to the API presented here. Another point that is missing in OGC Coverages API is that it does not provide a unified way to access all Coverage values in the same way for all Coverage types. We hope that this work will influence the existing OGC Coverage API in a way that it will become simpler to use.

An important aspect that was not discussed in the present paper is related to the definitions of functions which represent filtered version of existing ones. This is a very useful functionality, especially if existing functions need to be queried and e.g. visualized as a function with a smaller number of arguments (time series representing values of a single cell of a time-dependent regular grid). In many cases this filtered functions need to be stored somewhere next to the real functions. Another example is when variable represents an aggregated version of another variable. It becomes a very tricky task when we add a user requirement that the connection between

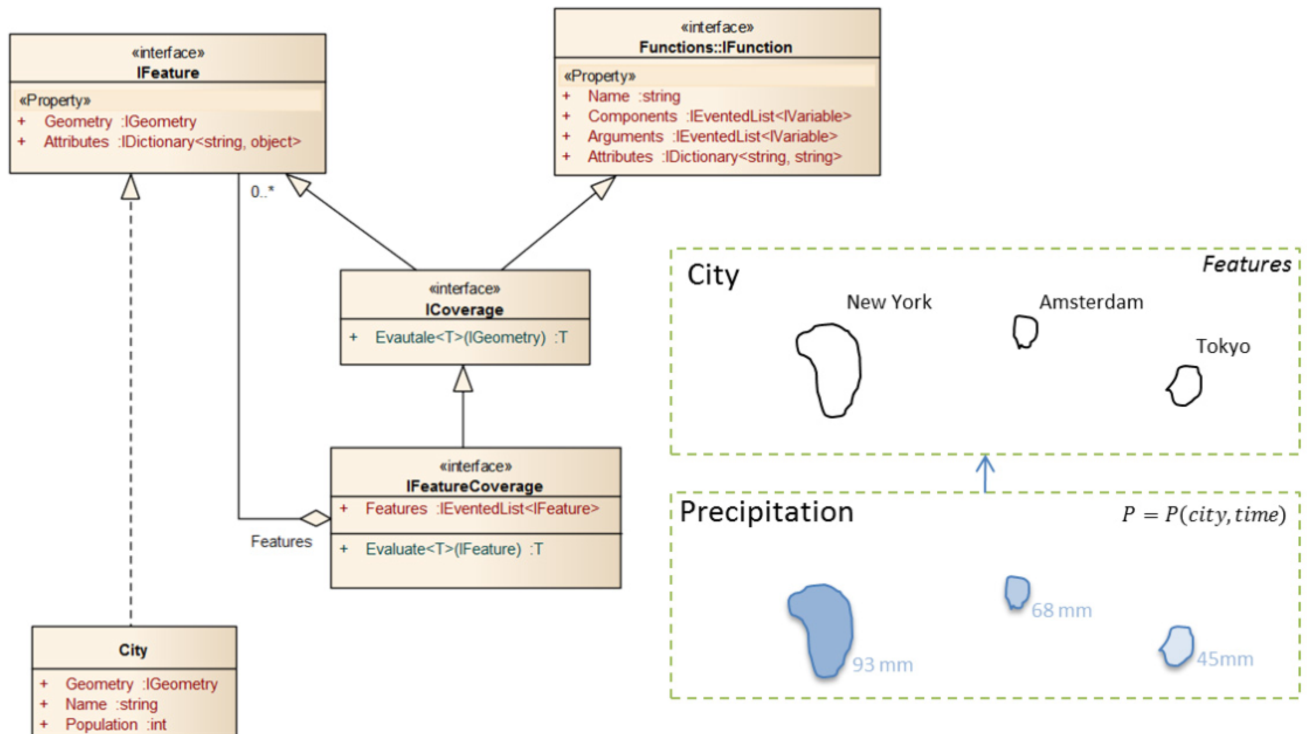


Figure 11: FeatureCoverage, function where one of the arguments uses features as values

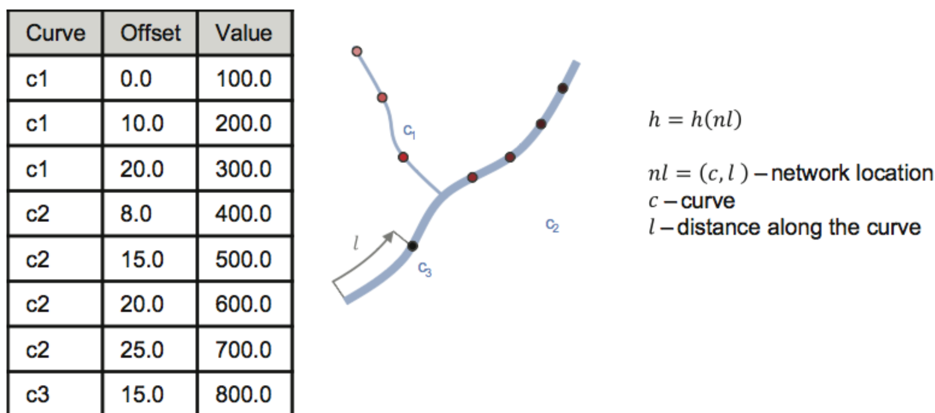
```
public class City : Feature
{
    [FeatureAttribute] public string Name { get; set; }
    [FeatureAttribute] public int Population { get; set; }
}

[Test]
[Category("Example")]
public void FeatureCoverage()
{
    var cities = new[]
                     {
                         new City { Name = "New York", Geometry = new Point(1.0, 1.0)} ,
                         new City { Name = "Amsterdam", Geometry = new Point(2.0, 2.0)} ,
                         new City { Name = "Tokyo", Geometry = new Point(3.0, 3.0)} ,
                     };

    // construct coverage
    var coverage = new FeatureCoverage { Features = cities, Name = "cities" };
    coverage.Arguments.Add(new Variable<City>("city"));
    coverage.Components.Add(new Variable<double>("precipitation"));

    // set values
    coverage[cities[0]] = 45.0;
    coverage[cities[1]] = 68.0;
    coverage[cities[2]] = 93.0;
}
```

Figure 12: Creation and use of FeatureCoverage using existing features (C#)

original and filtered variables must be live, meaning that when values in the original variable change - values in the filtered variable will be recalculated automatically.

7 Conclusion

We have presented an API that provides users the possibility to work with geospatial and non-geospatial types of multidimensional data in a convenient way. By providing a direct connection to the NetCDF data format we hope that our API will become especially popular for working with results from numerical models. The library implementation also fills in



Curve	Offset	Value
c1	0.0	100.0
c1	10.0	200.0
c1	20.0	300.0
c2	8.0	400.0
c2	15.0	500.0
c2	20.0	600.0
c2	25.0	700.0
c3	15.0	800.0

Figure 13: Network coverage, values defined along the network branches.

```python
# create simple network containing 4 nodes and 3 branches
network = HydroNetwork(Name = "test network")

n1 = Node(Name = "node1", Geometry = Point(10, 10))
n2 = Node(Name = "node2", Geometry = Point(20, 20))
n3 = Node(Name = "node3", Geometry = Point(30, 30))
n4 = Node(Name = "node4", Geometry = Point(40, 40))

c1 = Branch(Name = "branch1", Source = n1, Target = n2)
c2 = Branch(Name = "branch2", Source = n2, Target = n3)
c3 = Branch(Name = "branch2", Source = n4, Target = n3)

# add nodes and branches to the network
network.Nodes.AddRange((n1, n2, n3, n4))
network.Branches.AddRange((c1, c2, c3))

# create and fill values of the network coverage
networkCoverage = NetworkCoverage()
networkCoverage.Network = network

networkCoverage[NetworkLocation(c1,  0.0)] = 100.0
networkCoverage[NetworkLocation(c1, 10.0)] = 200.0
networkCoverage[NetworkLocation(c1, 20.0)] = 300.0
networkCoverage[NetworkLocation(c2,  8.0)] = 400.0
networkCoverage[NetworkLocation(c2, 15.0)] = 500.0
networkCoverage[NetworkLocation(c2, 20.0)] = 600.0
networkCoverage[NetworkLocation(c2, 25.0)] = 700.0
networkCoverage[NetworkLocation(c3, 15.0)] = 800.0
```

Figure 14: Create network and network coverage (Python)

the gap of a .NET based NetCDF API that makes use of the features of the .NET platform.

The design of an API often feels a bit like tightrope walking. There need to be a balance between high level of usability and performance on one side, as well as a balance between completeness versus simplicity.

By following the general guidelines of a domain driven design the API does adhere to good practices. Whether it is actually a usable one depends on the experience of users. Therefore we invite readers to try out the API and provide us with feedback and critical comments.

The API as well as its implementation will be released as an open-source project. Currently a draft version is already available as a branch of a SharpMap project.[37]

Bibliography

V. Balaji and Z. Liang. Gridspec: A standard for the description of grids used in earth system models. In *Workshop on Community Standards for Unstructured Grids*, October 2006.

G. Donchyts, S. Hummel, S. Vaneçek, J. Groos, A. Harper, R. Knapen, J. Gregersen, P. Schade, A. Antonello, and P. Gijsbers. Openmi 2.0 what's new? In *International Congress on Environmental Modelling and Software, July*, pages 5–8, 2010.

JL Goodall, DR Maidment, and J. Sorenson. Representation of spatial and temporal data in arcgis. *GIS and Water Resources III. AWRA, Nashville, TN*, 2004.

B. Howe and D. Maier. Algebraic manipulation of scientific datasets. *The VLDB journal*, 14(4):397–416, 2005.

H.W.J. Kernkamp, A. van Dam, G.S. Stelling, and E.D. de Goede. Efficient scheme for the shallow water equations on unstructured grids with application to the continental shelf. *Ocean Dynamics*, 61 (8):1175–1188, 2011. doi: doi:10.1007/s10236-011-0423-6.

S. Nativi, J. Caron, B. Domenico, and L. Bigagli. Unidata's common data model mapping to the iso 19123 data model. *Earth Science Informatics*, 1(2):59–78, September 2008. doi: 10.1007/s12145-008-0011-6.

B. Nordgren. An iso19123 coverage primer. Integration guide, USDA Forest Service, 2006.

D.J. Peuquet. Time in gis and geographical databases. *Geographical information systems*, 1:91–103, 1999.

[37] http://bit.ly/functional-coverages

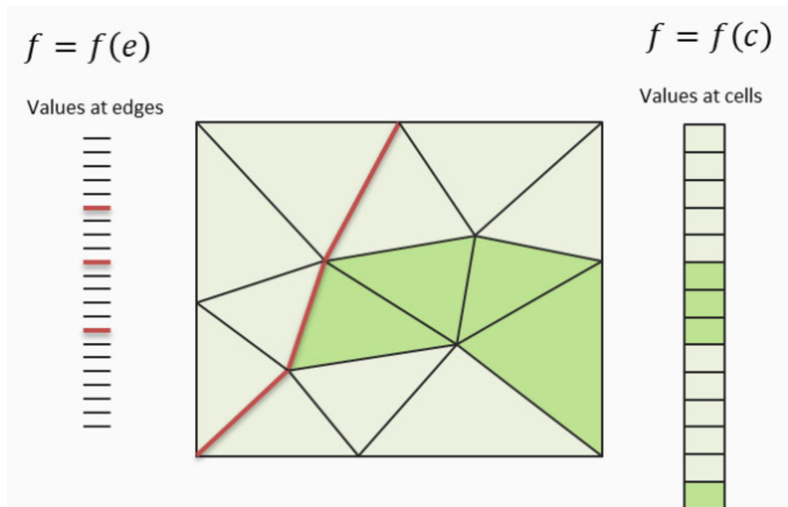


Figure 15: Values defined on unstructured grid cells (faces) or interfaces between cells (edges)

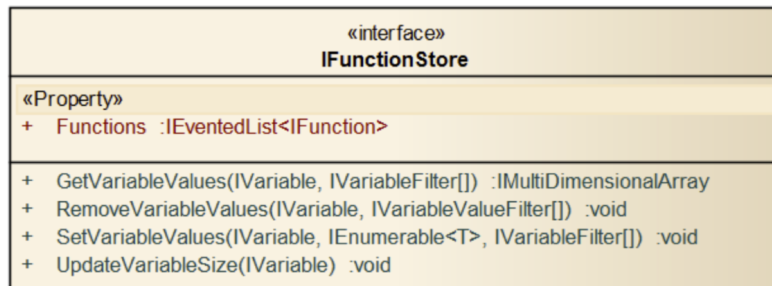


Figure 16: IFunctionStore - repository of functions and their values

R. Rew and G. Davis. Netcdf: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990.

G.K. Rutledge, J. Alpert, and W. Ebisuzaki. A climate and weather model archive at the national oceanic and atmospheric administration. *Bulletin of the American Meteorological Society*, 87:327–341, 2006. doi: 10.1175/BAMS-87-3-327.

W.J. Schroeder, L.S. Avila, and W. Hoffman. Visualizing with vtk: a tutorial. *Computer Graphics and Applications, IEEE*, 20(5):20–27, 2000.

R.P. Signell, S. Carniel, J. Chiggiato, I. Janekovic, J. Pullen, and C.R. Sherwood. Collaboration tools and techniques for large model datasets. *Journal of Marine Systems*, 69(1-2):154–161, 2008.

L. Treinish. A function-based data model for visualization. In *Proceedings of the IEEE Visualization 1999 Conference Late Breaking Hot Topics*, pages 73–76, 1999.

M. Wachowicz. *Object-oriented design for temporal GIS*. CRC, 1999.

Appendix: Acronyms

OGC Open Geospatial Consortium

NetCDF Network Common Data Format

UCAR University Corporation for Atmospheric Research

API Application Programming Interface

GDAL Geospatial Data Abstraction Library

CDM Common Data Model

OpenMI Open Modeling Interface

VTK Visualization Toolkit

GIS Geographic Information System Geographic information – Schema for coverage geometry and functions

Appendix: Definition of Terms

Feature Geospatial Feature as defined by OGC. We will call Feature any type that implements at least the Geometry property and has a set of other attributes.

Geometry Feature Geometry as defined by OGC. An attribute of the Feature.

Coverage Specific type of Feature that can generate a value for any point within its domain. Examples include raster images, a polygon overlay or a digital elevation matrix. In other words, it is a geospatial feature and a function at the same time.

Function A vector(-valued) function. A function of one or more variables whose range is n-dimensional. It associates dependent variable (component) values with independent variable (argument) values. As an example, a scalar function has $n = 1$ (a one-dimensional *range*-space), but it may still 'live' on a multi-dimensional *domain*-space.

Variable Defines a value that may change within the scope of a given problem. The mathematical definition is used, not to be confused with a computer science variable. Variables can be independent or dependent on other variables.

Unit Defines unit of measure for variable values. For example [m], [m/s], etc.

Array In general a multi-dimensional array of some value type.

Gennadii Donchyts, Fedor Bart, H.R.A (Bert) Jagers, Arthur van Dam:
Deltares,
Rotterdamseweg 185, Delft, The Netherlands
P.O. Box 177, 2600 MH Delft, The Netherlands
gennadii.donchyts@deltares.nl
fedor.baart@deltares.nl
bert.jagers@deltares.nl
arthur.vandam@deltares.nl

This PDF article file is a sub-set from the larger OSGeo Journal. For a complete set of articles please the Journal web-site at:

http://osgeo.org/journal

and ready for the future. TRLIB was only recently released under an open source license (and made available through https://bitbucket.org/KMS/trlib), but in the near future we hope to implement means for better interoperability with the more well established libraries in the open source geomatics field.

Acknowledgements: We thank Willy Lehmann Weng and Knud Poder for commenting on the draft of this paper.

Bibliography

C. W. Clenshaw. A note on the summation of chebyshev series. *Math. Tables Aids Comput.*, 9(51):118–120, 1955. URL http://www.jstor.org/stable/2002068.

Karsten E. Engsager and Knud Poder. A highly accurate world wide algorithm for the transverse mercator mapping (almost). In *Proc. XXIII Intl. Cartographic Conf. (ICC2007), Moscow*, page 2.1.2, 2007.

Gerald I. Evenden. Cartographic projection procedures for the unix environment—a user's manual, 1990. US Geological Survey Open-File Report 90–284.

C. Gram, O. Hestvik, H. Isakson, P.T. Jacobsen, J. Jensen, P. Naur, B.S. Petersen, and B. Svejgaard. Gier - a danish computer of medium size. *IEEE Transactions on Electronic Computers*, EC-12(5):629–650, December 1963. URL http://www.datamuseum.dk/site_dk/rc/gierdoc/ieeeartikel.pdf.

Charles Karney. Transverse mercator with an accuracy of a few nanometers. *Journal of Geodesy*, 2011(1):1–11, 2011. doi: http://dx.doi.org/10.1007/s00190-011-0445-3.

R. König and K. H. Weise. *Mathematische Grundlagen der Höheren Geodäsie und Kartographie, Erster Band*. Springer, Berlin/Göttingen/Heidelberg, 1951.

Knud Poder and Karsten Engsager. Some conformal mappings and transformations for geodesy and topographic cartography, 1998. National Survey and Cadastre, Denmark, Publications, 4. series vol. 6.

Carl Christian Tscherning and Knud Poder. Some geodetic applications of clenshaw summation. *Bolletino di Geodesia e Scienze Affini*, XLI(4):349–375, 1982.

Thomas Knudsen
National Survey and Cadastre
kms. dk
thokn AT kms DOT dk

Imprint - Volume 10

Editor in Chief:
Tyler Mitchell, Locate Press - info@locatepress.com

Proceedings Editor:
Gary Sherman, GeoApt LLC - gsherman@geoapt.com

FOSS4G Academic Committee:

Chair - Rafael Moreno Univ. of Colorado, Denver, USA
Thierry Badard Laval University, Canada
Maria Brovelli Politecnico di Milano campus Como, Italy
Serena Coetzee University of Pretoria, South Africa
Tyler Erickson Michigan Tech Research Institute, USA
Songnian Li Ryerson University, Canada
Jeff McKenna Geteway Geomatics, Canada
Helena Mitasova North Carolina State University, USA
Venkatesh Raghavan Osaka City University, Japan

Acknowledgements
Gary Sherman, LaTeX magic & layout support
Various reviewers & writers

The *OSGeo Journal* is a publication of the *OSGeo Foundation*. The base of this journal, the LaTeX 2ε style source has been kindly provided by the GRASS and R News editorial boards.

This work is licensed under the Creative Commons Attribution-No Derivative Works 3.0 License. To view a copy of this licence, visit: http://creativecommons.org/licenses/by-nd/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

All articles are copyrighted by the respective authors. Please use the OSGeo Journal url for submitting articles, more details concerning submission instructions can be found on the OSGeo homepage.

Journal online: http://www.osgeo.org/journal

OSGeo Homepage: http://www.osgeo.org

Mail contact through OSGeo, PO Box 4844, Williams Lake, British Columbia, Canada, V2G 2V8

ISSN 1994-1897