Finite Element based CF proposal for Unstructured Grid data model

This page describes a proposal for storing unstructured model data in a netCDF file. Compared to the other proposal, this proposal intends to not only store the mesh and the (staggered) values, but also the information required for a consistent spatial interpretation of the value. The additional information may for instance indicate whether values should be considered constant or linearly varying across elements. The concepts are described in an attached memo and some mails to the Google group on ugrid-interoperability by David Ham (ICOM model).

The proposal distinguishes the following concepts:

- vertices: these are the (abstract) points that form the basis of the mesh.
- topology: the topology defines the connectivity of the vertices and defines the elements.
- function space: a function space links elements to the indices of the degrees of freedom.
- fields: the fields are variables that specify a value for each degree of freedom.

Due to the wide variety in unstructured mesh models, some relevant concepts have not yet been worked out in detail. This includes the following concepts:

- adaptive mesh topology (this could be supported by defining a `time_concatenation` attribute for a time-series of mesh topologies)
- 3D fully unstructured meshes (this can be defined analogous to the conventions for 2D mixed element topologies)
- local model features, such as weirs and other structures located at cell interfaces (see e.g. the other other proposal)

More details can be found in the various sections below:

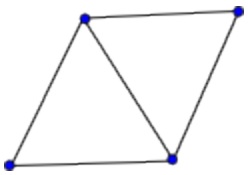
Topology.

The storage of the topology doesn't deviate much from the other proposal. However, for the time being we will keep the examples more lean and mean by indicating only the minimal data. Furthermore, one should note that where the other proposal uses the word `node` for the points of the mesh, this proposal uses the word `vertex` for those points.

2D triangular mesh topology.

The topology information is stored as attributes to a dummy variable (in the example below called "Topology"). The first new attribute is `dimensionality`: its value should be integer 2 for a 2D (triangular) mesh. The second new attribute is `element_vertex_connectivity` that specifies the actual topology of how the (triangular) elements map to the vertices. Note that this maps to the `face_node_connectivity` in the other proposal. The `vertex_coordinates` attribute points to the lat,lon coordinates of the vertices; we will fill in the details below once we have defined the variables on the topology.

Consistent with the CF-conventions compression option, the connectivity indices are 0-based, i.e. if the "Element_vertices" array contains values 0, 1 and 2 then this means that that element connects the first three points.



Example:

```
dimensions:
        nElements = 2 ;
        Three = 3 ;

variables:
// Mesh topology
        integer Element_vertices(nElements, Three) ;
                Element_vertices:standard_name = "" ;  // YET TO BE DETERMINED
                Element_vertices:long_name = "Maps every triangular element to its three corner vertices." ;
        integer Topology ;
                Topology:standard_name = "" ;  // YET TO BE DETERMINED
                Topology:long_name = "Topology data of 2D unstructured mesh" ;
                Topology:dimensionality = 2 ;
                Topology:element_vertex_connectivity = "Element_vertices" ;
                Topology:vertex_coordinates = "...see below..." ;
```

The example explicitly defines a simple topology consisting of two triangles. This example also implicitly defines a second topology, namely the topology of the boundary edges on which boundary forcings may be prescribed. If this second topology is going to be used, then it's probably wise to make this definition more explicit.

Function Space.

The next step is defining the function space on the mesh. One needs to define the basis functions that span up the function space and the element topology on which they are defined. For this we define function space variables as integer variables that via its values points for every element to the associated degrees of freedom of the function space, and via its attribute `standard_basis_functions` points to the basis functions associated with this function space (here: the linear basis functions P1). Basis functions can be specified in two ways: either by means of the attribute `standard_basis_functions` in which case its value is taken from the table of commonly used function spaces below.

Name	Meaning
P0	constant functions on element
P1	linear continuous functions
...	...

The second way to define the basis functions is by explicitly specifying the equations for the basis functions by means of an attribute `basis_functions_formula`. For more information see the section on basis functions below.

Consistent with the CF-conventions compression option, the function space indices are 0-based, i.e. if the "FSpace" array contains values 0, 1 and 2 then this means that that element is associated with the first three degrees of freedom.

Example:

```
dimensions:
        nElements = 2 ;
        Three = 3 ;
        nDoFperElement = 3 ;

variables:
// Mesh topology
        integer Element_vertices(nElements, Three) ;
                Element_vertices:standard_name = "" ;  // YET TO BE DETERMINED
                Element_vertices:long_name = "Maps every triangular element to its three corner vertices." ;
        integer Topology ;
                Topology:standard_name = "" ;  // YET TO BE DETERMINED
                Topology:long_name = "Topology data of 2D unstructured mesh" ;
                Topology:dimensionality = 2 ;
                Topology:element_vertex_connectivity = "Element_vertices" ;
                Topology:vertex_coordinates = "...see below..." ;

// Function spaces
        integer FSpace(nElements, nDoFperElement) ;
                FSpace:standard_name = "" ; // YET TO BE DETERMINED
                FSpace:long_name = "Function space S1: maps every triangular element to the associated degrees-
of-freedom." ;
                FSpace:standard_basis_functions = "P1" ;
```

For a continuous function space the degrees of freedom for the individual elements will be linked. In that case the FSpace variables contains the same data as the Element_vertices table (and the total number of freedoms will equal the number of vertices). For a discontinuous function space this will not be the case (and the total number of freedoms will equal three times the number of elements).

Field Variables.

The final step is in defining the actual variable fields (i.e. the values associated with the degrees of freedom of the function space). This is done by specifying variables of dimension nDegreesOfFreedom (constant in time) or by variables of dimension (nTimeSteps,nDegreesOfFreedom) where nTimeSteps matches the dimensionality of the time coordinate and nDegreesOfFreedom matches the number of degrees of freedom associated with the function space referred to by the variable attribute `function_space`.

Example:

```
dimensions:
        nElements = 2 ;
        nDoFperElement = 3 ;
        nDoFofFSpace = 4 ;

        // common dimensions
        Three = 3 ;
        time = UNLIMITED ; // (1 currently)

variables:
// Mesh topology
        integer Element_vertices(nElements, Three) ;
                Element_vertices:standard_name = "" ;  // YET TO BE DETERMINED
                Element_vertices:long_name = "Maps every triangular element to its three corner vertices." ;
        integer Topology ;
                Topology:standard_name = "" ;  // YET TO BE DETERMINED
                Topology:long_name = "Topology data of 2D unstructured mesh" ;
                Topology:dimensionality = 2 ;
                Topology:element_vertex_connectivity = "Element_vertices" ;
                Topology:vertex_coordinates = "lat lon" ;

// Function spaces
        integer FSpace(nElements, nDoFperElement) ;
                FSpace:standard_name = "" ; // YET TO BE DETERMINED
                FSpace:long_name = "Function space S1: maps every triangular element to the associated degrees-
of-freedom." ;
                FSpace:standard_basis_functions = "P1" ;

// Time coordinate
        double time(time) ;
                time:standard_name = "time" ;
                time:units = "seconds since 1992-08-31 00:00:00" ;

// Variables
        double zwl(time, nDoFofFSpace) ;
                zwl:standard_name = "sea_surface_height_above_geoid" ;
                zwl:units = "m" ;
                zwl:function_space = "FSpace" ;
        double depth(nDoFofFSpace) ;
                depth:standard_name = "sea_floor_depth_below_geoid" ;
                depth:units = "m" ;
                depth:positive = "down" ;
                depth:function_space = "FSpace" ;
        double lat(nDoFofFSpace) ;
                lat:standard_name = "latitude" ;
                lat:units = "degrees_east" ;
                lat:function_space = "FSpace" ;
        double lon(nDoFofFSpace) ;
                lat:standard_name = "longitude" ;
                lat:units = "degrees_north" ;
                lat:function_space = "FSpace" ;
```

The example shows water level, bed level and the coordinates for a case with all linear basis functions. More complex examples need to be added.

Defining basis function formulae.

Below you'll find an example of how basis functions could be specified in netCDF:

```
dimensions:
    Three = 3 ;
    maxStringLen = 5 ;

variables:

// Degree of freedom locations in local coordinates.
    double DOF_locations_P1_Tri(Three, Three) ;
        DOF_locations_P1_Tri:standard_name = "" ; // YET TO BE DETERMINED
        DOF_locations_P1_Tri:long_name = "DOF locations in local coordinates for P1 Triangular elements"
        DOF_locations_P1_Tri:dimension = 2 ;
        DOF_locations_P1_Tri:coordinates = "barycentric" ;


// Basis functions
    char Basis_functions_P1_Tri(Three,maxStringLen) ;
        Basis_functions_P1_Tri:standard_name = "" ; // YET TO BE DETERMINED
        Basis_functions_P1_Tri:long_name = "Basis functions in local coordinates for P1 Triangular elements" ;
        Basis_functions_P1_Tri:continuity = "continuous";
        Basis_functions_P1_Tri:topological_dimension = 2 ;
        Basis_functions_P1_Tri:data_dimension = 1 ;
        Basis_functions_P1_Tri:coordinates = "barycentric" ;



data:

    DOF_locations_P1_Tri = 1, 0, 0,
                           0, 1, 0,
                           0, 0, 1;

    Basis_functions_P1_Tri = "xi[0]",
                             "xi[1]",
                             "xi[2]";
```