Common Data Model
Scientific Feature Types

John Caron
UCAR/Unidata
July 8, 2008
Contents

• Overview / Related Work
• CDM Feature types (focus on point data)
• Nested Table notation for Point Features
• Representing Point Data in Netcdf-3/CF
  – Preliminary experiments with BUFR
Unidata’s Common Data Model

- Abstract data model for scientific data
- NetCDF-Java library implementation/prototype
- Features are being pushed into the netCDF-4 C library
Common Data Model

Scientific Feature Types
- Point
- Trajectory
- Station
- Profile
  - Radial
  - Grid
  - Swath

Coordinate Systems

Data Access
- netCDF-3, HDF5, OPeNDAP
- BUFR, GRIB1, GRIB2, NEXRAD, NIDS, McIDAS, GEMPAK, GINI, DMSP, HDF4, HDF-EOS, DORADE, GTOPO, ASCII
Related Standards/Models

• National and International committees are mandating compliance with ISO/OGC data standards
• Where does the CDM fit in?
ISO/TC 211 Reference model

OGC Abstract Specification

OGC WXS = Web (MFC) Service client/server protocols

GML encoding

CSML

ncML-Gml

Abstract

netCDF-3, HDF5, OPeNDAP, BUFR, GRIB1, GRIB2, NEXRAD, NIDS, McIDAS, GEMPAK, GINI, DMSP, HDF4, HDF-EOS, DORADE, GTOPO, ASCII
Where does CDM fit?

- Bridge between actual datasets and abstract data model(s)
- Translate file’s “native data model” into higher-level semantic model
- “bottom-up” vs “top-down” approach
Climate Science Modelling Language (CSML)

- British Atmospheric Data Center (BADC)
- Uses ISO/OGC semantic model
- GML application schema for atmospheric and oceanographic data
## CSML - CDM Feature types

<table>
<thead>
<tr>
<th>CSML Feature Type</th>
<th>CDM Feature Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointFeature</td>
<td>PointFeature</td>
</tr>
<tr>
<td>PointSeriesFeature</td>
<td>StationFeature</td>
</tr>
<tr>
<td>TrajectoryFeature</td>
<td>TrajectoryFeature</td>
</tr>
<tr>
<td>PointCollectionFeature</td>
<td>StationFeature at fixed time</td>
</tr>
<tr>
<td>ProfileFeature</td>
<td>ProfileFeature</td>
</tr>
<tr>
<td>ProfileSeriesFeature</td>
<td>StationProfileFeature at one location and fixed vertical levels</td>
</tr>
<tr>
<td>RaggedProfileSeriesFeature</td>
<td>StationProfileFeature at one location</td>
</tr>
<tr>
<td>SectionFeature</td>
<td>SectionFeature with fixed number of vertical levels</td>
</tr>
<tr>
<td>RaggedSectionFeature</td>
<td>SectionFeature</td>
</tr>
<tr>
<td>ScanningRadarFeature</td>
<td>RadialFeature</td>
</tr>
<tr>
<td>GridFeature</td>
<td>GridFeature at a single time</td>
</tr>
<tr>
<td>GridSeriesFeature</td>
<td>GridFeature</td>
</tr>
<tr>
<td>SwathFeature</td>
<td>SwathFeature</td>
</tr>
</tbody>
</table>
You were there
CDM Feature Types

Formerly known as Scientific Data Types
• Based on examining real datasets “in the wild”
  – Attempt to categorize, so that datasets can be handled in a more general way
• Implementation for OGC feature services
  – Intended to scale to large, multifile collections
  – Intended to support “specialized queries”
    • Space, Time
• Data abstraction
  – Netcdf-Java has prototype implementation
Gridded Data

- **Grid**: multidimensional grid, separable coordinates
- **Radial**: a connected set of *radials* using polar coordinates collected into *sweeps*
- **Swath**: a two dimensional grid, *track* and *cross-track* coordinates
Gridded Data

- Cartesian coordinates
- Data is 2, 3, 4D
- All dimensions have 1D coordinate variables (separable)

```c
float gridData(t, z, y, x);
float t(t);
float y(y);
float x(x);
float z(z);
```

```c
float lat(y, x);
float lon(y, x);
float height(t, z, y, x);
```
Radial Data

- Polar coordinates
- two dimensional
- Not separate time dimension

float radialData(radial, gate) :
  float distance(gate)
  float azimuth(radial)
  float elevation(radial)
  float time(radial)

float origin_lat;
float origin_lon;
float origin_alt;
Swath

- two dimensional
- track and cross-track
- not separate time dimension
- orbit tracking allows fast search

float `swathData(track, xtrack)`
float `lat(track, xtrack)`
float `lon(track, xtrack)`
float `alt(track, xtrack)`
float `time(track)"
Unstructured Grid

- Pt dimension not connected
- Need to specify the connectivity explicitly
- No implementation in the CDM yet

```c
float unstructGrid(t, z, pt);
float lat(pt);
float lon(pt);
float time(t);
float height(z);
```
Be here now
1D Feature Types ("point data")

`float data(sample);`

- **Point**: measured at one point in time and space
- **Station**: time-series of points at the same location
- **Profile**: points along a vertical line
- **Station Profile**: a time-series of profiles at same location.
- **Trajectory**: points along a 1D curve in time/space
- **Section**: a collection of profile features which originate along a trajectory.
Point Observation Data

• Set of measurements at the same point in space and time = obs
• Collection of obs = dataset
• Sample dimension not connected

float obs1(sample);
float obs2(sample);
float lat(sample);
float lon(sample);
float z(sample);
float time(sample);

Table {
  lat, lon, z, time;
  obs1, obs2, ...
} obs(sample);
Time-series Station Data

float obs1(sample);
float obs2(sample);
float lat(sample);
float lon(sample);
float z(sample);
float time(sample);

float obs1(stn, time);
float obs2(stn, time);
float time(stn, time);

int stationId(stn);
float lat(stn);
float lon(stn);
float z(stn);

float obs1(stn, time);
float obs2(stn);
int stn_id(sample);
float time(sample);

int stationId(stn);
float lat(stn);
float lon(stn);
float z(stn);

Table {
    stationId;
    lat, lon, z;
} obs(*); // connected
} stn(stn); // not connected
Profile Data

float obs1(sample);
float obs2(sample);
float lat(sample);
float lon(sample);
float z(sample);
float time(sample);

float obs1(profile, level);
float obs2(profile, level);
int profile_id(sample);
float z(sample);

float time(profile);
float lat(profile);
float lon(profile);

int profileId(profile);
float lat(profile);
float lon(profile);
float time(profile);

Table {
  profileId;
  lat, lon, time;
} profile(profile); // not connected

Table {
  profileId;
  lat, lon, time;
  z;
  obs1, obs2, ...
} obs(*); // connected
Time-series Profile Station Data

float obs1(profile, level);
float obs2(profile, level);
float z(profile, level);

float time(profile);
float lat(profile);
float lon(profile);

float obs1(stn, time, level);
float obs2(stn, time, level);
float z(stn, time, level);

float time(stn, time);
float lat(stn);
float lon(stn);

Table {
  stationId;
  lat, lon;
Table {
  time;
Table {
    z;
    obs1, obs2, ...
  } obs(*); // connected
} profile(*); // connected
} stn(stn); // not connected
Trajectory Data

<table>
<thead>
<tr>
<th>Float</th>
<th>Sample</th>
</tr>
</thead>
<tbody>
<tr>
<td>obs1</td>
<td></td>
</tr>
<tr>
<td>obs2</td>
<td></td>
</tr>
<tr>
<td>lat</td>
<td></td>
</tr>
<tr>
<td>lon</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td></td>
</tr>
</tbody>
</table>

```
float obs1(sample);
float obs2(sample);
float lat(sample);
float lon(sample);
float z(sample);
float time(sample);
int trajectory_id(sample);
```
Section Data

float obs1(traj, profile, level);
float obs2(traj, profile, level);
float z(traj, profile, level);
float lat(traj, profile);
float lon(traj, profile);
float time(traj, profile);

Table {
  section_id;
  Table {
    surface_obs // data anywhere
    lat, lon, time
    Table {
      depth;
      obs1, obs2, ...
    } obs(*); // connected
  } profile(*); // connected
  } section(*) // not connected
Nested Table Notation (1)

1. A *feature instance* is a row in a table.
2. A *table* is a collection of features of the same type. The table may be fixed or variable length.
3. A nested (*child*) table is owned by a row in the *parent* table.
4. Both *coordinates* and *data variables* can be at any level of the nesting.
5. A *feature type* is represented as nested tables of specific form.
6. A *feature collection* is an unconnected collection of a specific feature type.
Nested Table Notation (2)

- A constant coordinate can be factored out to the top level. This is logically joined to any nested table with the same dimension.

```plaintext
dim level = 17;
float z(level);

Table {
    data1, data2
    lat, lon, time;
}

Table {
    obs1, obs2, ...
} obs(level);

} profile(*);
```
Nested Table Notation (3)

- A coordinate in an inner table is connected; a coordinate in the outermost table is unconnected.

```
Table {
  trajectory_id;
  Table {
    lat, lon, z, time;
    Table {
      obs1, obs2, ...
    } obs(*); // connected
  } traj(traj) // not connected
} point(sample);
```

```
Table {
  stationId;
  lat, lon;
  Table {
    time;
    Table {
      z;
      obs1, obs2, ...
    } obs(*); // connected
    } profile(*); // connected
  } stn(stn); // not connected
```
Relational model

• Nested Tables are a hierarchical data model (tree structure)

• Simple transformation to relational model – explicitly add join variables to tables

```plaintext
Table {
    stationId;
    lat, lon, z;

    Table {
        time;
        obs1, obs2, ...
    } obs(42);
}

RTable {
    stationId // primary key
    lat, lon, z;
}

RTable {
    stationId // secondary key
    time;
    obs1, obs2, ...
}

obs;
```
Nested Model Summary

• Compact notation to describe 1D point feature types
  – Connectivity of points is key property
  – Variable/fixed length table dimensions can be notated easily
  – Constant/varying coordinates can be easily seen

• Can be translated to relational model to get different performance tradeoffs
Representing point data in netCDF3/CF (or) Fitting data into unnatural shapes
Representing point data in NetCDF-3 / CF

• Many existing files already store point data in netCDF-3, but not standardized.
• CF Convention has 2 simple examples, no guidance for more complex situations
• Can use Nested Tables as comprehensive abstract model of data
• Look for general solutions
CF Example 1: Trajectory data

float O3(time) ;
    O3:coordinates = "time lon lat z" ;
double time(time) ;
float lon(time) ;
float lat(time) ;
float z(time) ;

Problem: what if multiple trajectories in same file?
CF Example 2: Station data

```c
float data(time, station);
  data:coordinates = "lat lon alt time" ;
double time(time) ;
float lon(station) ;
float lat(station) ;
float alt(station) ;
```

If stations have different times, use
```c
double time(time, station) ;
```
Problem: what if stations have different number of times?
Rectangular Array (netCDF-3)

A standard two-dimensional array is a rectangle.

Ragged Array

With Variant arrays, you need not waste space.
Storing Ragged Arrays

**Rectangularize** the Array: use maximum size of the ragged array, use missing values
- Works well if avg ~ max
- Or if you will store/transmit compressed

**Linearize** the Array: put all elements of the ragged array into a 1D array
1. Connect using index ranges
2. Connect using linked lists
3. Connect by matching field values (relational)
4. Index join
Linearize Ragged Arrays
Index Ranges
Linearize Ragged Arrays
Linked List of Indices

Diagram showing a linked list structure with 'Parent' and 'Child' connections.
Linearize Ragged Arrays
Match field values (relational)

<table>
<thead>
<tr>
<th>Lat</th>
<th>Lon</th>
<th>Alt</th>
<th>Stn</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.4</td>
<td>40.2</td>
<td>1033</td>
<td>KBO</td>
</tr>
<tr>
<td>77.2</td>
<td>-123</td>
<td>343</td>
<td>KFRC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stn</th>
<th>Time</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>KBO</td>
<td>12:05</td>
<td>32.8</td>
</tr>
<tr>
<td>KFRC</td>
<td>12:08</td>
<td>33.2</td>
</tr>
<tr>
<td>KFRC</td>
<td>12:13</td>
<td>28.9</td>
</tr>
<tr>
<td>KBO</td>
<td>12:13</td>
<td>33.8</td>
</tr>
<tr>
<td>KFRC</td>
<td>12:16</td>
<td>27.9</td>
</tr>
<tr>
<td>KFRC</td>
<td>12:19</td>
<td>19.9</td>
</tr>
<tr>
<td>KFRC</td>
<td>12:24</td>
<td>20.8</td>
</tr>
<tr>
<td>KBO</td>
<td>12:30</td>
<td>34.5</td>
</tr>
</tbody>
</table>
Linearize Ragged Arrays
Index Join

<table>
<thead>
<tr>
<th>Lat</th>
<th>Lon</th>
<th>Alt</th>
<th>Stn</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.4</td>
<td>40.2</td>
<td>1033</td>
<td>KBO</td>
</tr>
<tr>
<td>77.2</td>
<td>-123</td>
<td>343</td>
<td>KFRC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parent</th>
<th>Time</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12:05</td>
<td>32.8</td>
</tr>
<tr>
<td>2</td>
<td>12:08</td>
<td>33.2</td>
</tr>
<tr>
<td>2</td>
<td>12:13</td>
<td>28.9</td>
</tr>
<tr>
<td>1</td>
<td>12:13</td>
<td>33.8</td>
</tr>
<tr>
<td>2</td>
<td>12:16</td>
<td>27.9</td>
</tr>
<tr>
<td>2</td>
<td>12:19</td>
<td>19.9</td>
</tr>
<tr>
<td>2</td>
<td>12:24</td>
<td>20.8</td>
</tr>
<tr>
<td>1</td>
<td>12:30</td>
<td>34.5</td>
</tr>
</tbody>
</table>
### Nested Model ➔ netCDF

#### 1. Nested Table ➔ Pseudo-Structures

**Table**
- profileId;
- lat, lon, time;
- Table {
  - z;
  - obs1, obs2, ...
} obs(*);
- } profile(profile);

**Dimensions**
- profile = 42;
- obs = 714;

**Variables**
- int profileId(profile);
- float lat(profile);
- float lon(profile);
- float time(profile);
- float z(obs);
- float obs1(obs);
- float obs2(obs);
Storing Ragged Arrays

**Multidimensional / Rectangular**
- dimensions:
  - profile = 42;
  - levels = 17;
- variables:
  - float lat(profile);
  - float lon(profile);
  - float time(profile);
  - float z(profile, level);
  - float obs1(profile, level);
  - float obs2(profile, level);

**Relational**
- dimensions:
  - profile = 42;
  - obs = 2781;
- variables:
  - int profileId(profile);
  - float lat(profile);
  - float lon(profile);
  - float time(profile);
  - float z(obs);
  - float obs1(obs);
  - float obs2(obs);

**Index Join**
- dimensions:
  - profile = 42;
  - obs = 2781;
- variables:
  - float lat(profile);
  - float lon(profile);
  - float time(profile);
  - float z(obs);
  - float obs1(obs);
  - float obs2(obs);
  - int profileIndex(obs)
Storing Ragged Arrays

<table>
<thead>
<tr>
<th>Index Range:</th>
<th>Linked List:</th>
<th>Link + Parent:</th>
</tr>
</thead>
<tbody>
<tr>
<td>dimensions:</td>
<td>dimensions:</td>
<td>dimensions:</td>
</tr>
<tr>
<td>profile = 42;</td>
<td>profile = 42;</td>
<td>profile = 42;</td>
</tr>
<tr>
<td>obs = 2781;</td>
<td>obs = 2781;</td>
<td>obs = 2781;</td>
</tr>
<tr>
<td>variables:</td>
<td>variables:</td>
<td>variables:</td>
</tr>
<tr>
<td>float lat(profile);</td>
<td>float lat(profile);</td>
<td>float lat(profile);</td>
</tr>
<tr>
<td>float lon(profile);</td>
<td>float lon(profile);</td>
<td>float lon(profile);</td>
</tr>
<tr>
<td>float time(profile);</td>
<td>float time(profile);</td>
<td>float time(profile);</td>
</tr>
<tr>
<td>int firstObs(profile)</td>
<td>int firstObs(profile)</td>
<td>int firstObs(profile)</td>
</tr>
<tr>
<td>int numObs(profile)</td>
<td></td>
<td>int profileIndex(obs)</td>
</tr>
<tr>
<td>float z(obs);</td>
<td>float z(obs);</td>
<td>float z(obs);</td>
</tr>
<tr>
<td>float obs1(obs);</td>
<td>float obs1(obs);</td>
<td>float obs1(obs);</td>
</tr>
<tr>
<td>float obs2(obs);</td>
<td>float obs2(obs);</td>
<td>float obs2(obs);</td>
</tr>
<tr>
<td></td>
<td>int nextChild(obs)</td>
<td>int nextChild(obs)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>int profileIndex(obs)</td>
</tr>
</tbody>
</table>
Case Study: BUFR

- WMO standard for binary point data
- Table driven
- Variable length
- Motherlode/IDD feed
  - 150K messages, 5.5M obs, 1 Gbyte per day
  - 350 categories of WMO headers
  - 70 distinct BUFR types
BUFR \(\Rightarrow\) netCDF-3

- BUFR data is stored as unsigned ints
  - scale/offset/bit widths stored in external tables
  - bit packed
  - Variable-length arrays of data
- Translate to netCDF:
  - Align data on byte boundaries
  - Use standard scale/offset attributes
  - rectangularize or linearize ragged arrays
Profiler BUFR data
uncompressed, variable # of levels

<table>
<thead>
<tr>
<th></th>
<th>Size(Kb)</th>
<th>Zipped</th>
<th>ratio raw</th>
<th>ratio zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUFR</td>
<td>79.7</td>
<td>22.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetCDF multidim</td>
<td>104.6</td>
<td>17.9</td>
<td>1.3</td>
<td>.81</td>
</tr>
<tr>
<td>netCDF linear</td>
<td>95.0</td>
<td>17.7</td>
<td>1.2</td>
<td>.80</td>
</tr>
</tbody>
</table>
## Compressed BUFR data

**fixed length nested tables**

<table>
<thead>
<tr>
<th></th>
<th>EUMETSAT, single level upper air</th>
<th>NCEP, satellite sounding</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15 messages, 430 obs/message</td>
<td>73 messages, 60 obs/message</td>
</tr>
<tr>
<td><strong>BUFR</strong></td>
<td>Size: 173 Kb, Zip: 152 Kb</td>
<td>Size: 1291 Kb, Zip: 1227 Kb</td>
</tr>
<tr>
<td><strong>NetCDF</strong></td>
<td>Size: 1914 Kb, Zip: 145 Kb</td>
<td>Size: 3550 Kb, Zip: 1749 Kb</td>
</tr>
<tr>
<td><strong>ratio</strong></td>
<td>11:1.95</td>
<td>2.75:1.42</td>
</tr>
</tbody>
</table>
Point data in netCDF-3

Summary

• Main problem is ragged arrays
• Tradeoffs
  – ease-of-writing vs. ease-of-reading
  – storage size
  – More studies with BUFR data
• NetCDF-4 is likely straightforward, since it has variable length Structures
• CF proposal Real Soon Now
NetCDF-Java library 4.0
Point Feature API

• NetCDF-Java library 4.0 will have a new API based on Nested Table model
• New Sequence data type = variable length array of Structures
  – Iterators over StructureData objects
• Experimenting with:
  – Automatic analysis of datasets to guess feature type
  – Annotate/configure Feature Dataset to identify nested tables and coordinates (push into NcML?)
  – NcML aggregation over feature collections (?)
Conclusions

• CDM Feature Type model and implementation are evolving
• Nested Table notation provides a flexible way to characterize 1D point datasets
• Netcdf-Java 4.0 library has refactored point data implementation
• TDS will eventually provide new point subsetting services
Recent new documents

CDM Feature Types
CDM Point Feature Types

http://www.unidata.ucar.edu/software/netcdf-java/CDM/

Feedback:
– caron@ucar.edu