Unidata

Providing data services, tools, & cyberinfrastructure leadership that advance Earth system science, enhance educational opportunities, & broaden participation

Adapting Software to NetCDF's Enhanced Data Model

Russ Rew
UCAR Unidata

EGU, May 2010

UCAR COMMUNITY PROGRAMS

Overview

- Background
  - What is netCDF?
  - What is the netCDF “classic data model”?
  - What is the netCDF “enhanced data model”?
- Issues in upgrading
  - Why upgrade?
  - Why wait?
  - What is a “chicken-and-egg logjam”?
- Experience so far
- Concluding remarks

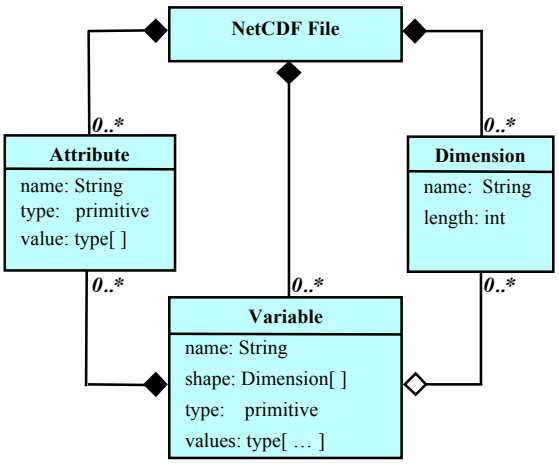
What is netCDF?

Development milestones

- **1989**: portable, self-describing data format, data model, and software for creation, access, and sharing of scientific data
- **1990's**: widespread use in ocean and climate modeling
- **2002**: Java version with OPeNDAP client support
- **2003**: NASA funded netCDF-4/HDF project; Argonne/Northwestern parallel netCDF
- **2004**: netCDF-Java plugins for reading other formats, NcML aggregation service
- **2007**: netCDF-Java Common Data Model
- **2008**: netCDF-4 C and Fortran library with HDF5 integration, ***enhanced data model***, parallel I/O
- **2009**: netCDF format standard endorsed by NASA
- **2010**: OPeNDAP client support for C/Fortran libraries; udunits, libcf, GridSpec libraries included

The netCDF classic data model

- A netCDF **File** has
  - **Variables**
  - **Dimensions**
  - **Attributes**
- Variables have
  - Name, shape, type, values
  - Associated attributes
- Dimensions have
  - Name, length
  - One dimension may be dynamic
- Variables may share dimensions
  - Indicates common grid
  - Scalar variables have no dimensions
- Primitive types
  - Numeric: byte. short, int, float, double
  - Character arrays for text

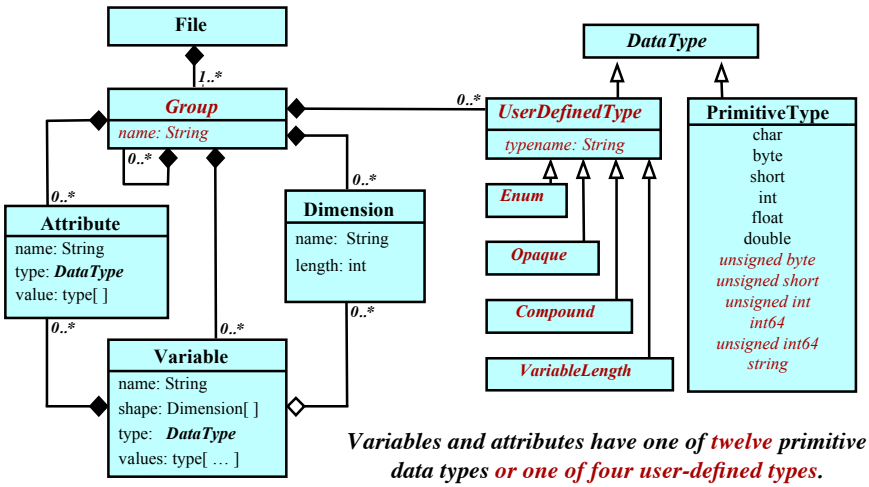


Evaluation: netCDF classic data model

- Strengths
  - Simple to understand and explain
  - Efficient reference implementation
  - Generic applications easy to develop
  - Good representations for gridded data
  - Shared dimensions useful for simple coordinate system representations
- Limitations
  - Small set of primitive types
  - Flat name space for naming data
  - Data structures limited to multidimensional arrays
  - Lacks compound structures, variable-length types, nested types, ragged arrays, enumerations

The netCDF-4 *enhanced* data model

A file has a top-level unnamed group. Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One or more dimensions may be of unlimited length.



Variables and attributes have one of twelve primitive data types or one of four user-defined types.

Evaluation: netCDF enhanced data model

- Strengths
  - Simpler than HDF5, with similar representational power
  - Compatible with existing data, software, conventions
  - Efficient reference implementation
  - Orthogonal features permit incremental adoption
- Limitations
  - More complex than classic data model
  - More challenging to develop general software tools
  - Comprehensive conventions still lacking
  - Not yet widely adopted

Why upgrade? Benefits of enhanced netCDF data model

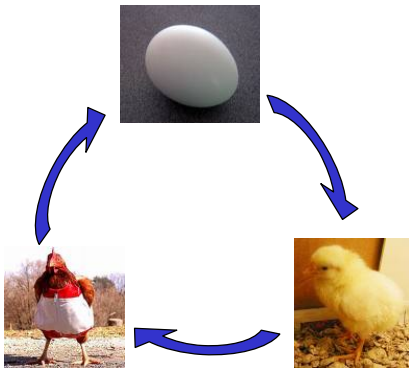
- More natural representations using
  - Strings and unsigned integer types
  - Nested data structures
  - Multiple unlimited dimensions and variable-length types
  - Ragged arrays
  - Hierarchical data organizations and name spaces
  - Enumerations
- Observational data using nested compound and variable-length types, e.g.

  Observations along ocean tracks; each track has a string ID, a string description, and a variable-length list of profiles; each profile has a latitude, longitude, time, and a variable-length list of observations; each observation records pressure, temperature, and salinity at various depths
- Ability to read other kinds of data through netCDF API
  - HDF-EOS, HDF4, HDF5, relational data, …

Why wait? Reasons to stick with classic netCDF model

- Combination of classic data model with netCDF-4
  - Only requires relinking instead of modifying software
  - Performance benefits: compression, multi-dimensional chunking, larger variables
- Data using enhanced data model not common yet
- Best practices and conventions not yet developed for enhanced data model
- NetCDF-4 enhanced data model not endorsed as a standard yet
- Developer perceptions
  - Must upgrade features of enhanced model all at once
  - Handling potentially infinite number of user-defined types is hard

Game of chicken: Who goes first?



- Data producers
  - Waiting until netCDF enhanced data model features are supported by more software, development of conventions
- Developers
  - Waiting for netCDF data that requires enhanced model and for development of conventions
- Convention creators
  - Waiting for data providers and software developers to identify needs for new conventions based on usage experience
- Result: “chicken-and-egg logjam”
  - *Delays effective use of advances in scientific data models for large and complex collections*

Experience so far: Adapting to netCDF-4

Features	NCAR's NCL	NetCDF Operators (NCO)	netCDF-Java	Python API	CCFE's C++ API for netCDF-4	ncdump ncgen nccopy
Performance features: compression, chunking, …	read-only	yes	read-only	yes	yes	yes
New primitive types	yes	yes	read-only	yes	yes	yes
Multiple unlimited dimensions	read-only	read-only	read-only	yes	yes	yes
Groups	not yet	not yet	read-only	yes	yes	yes
Compound types, variable-length types	not yet	not yet	read-only	flat	yes	yes

Experience developing nccopy utility

- Demonstrates any netCDF-4 data can be accessed through interface without previous or built-in knowledge of user-defined data types
- Showed netCDF-4 API is adequate for handling arbitrary nesting of groups and user-defined types
- Provides evidence that programming generic netCDF-4 applications is not too difficult
  - Classic data model: 494 lines of C
  - Enhanced data model: 911 lines of C
- Also demonstrates usefulness of additional higher-level APIs for tool developers
  - Iterator APIs for simpler data access
  - APIs that make recursion unnecessary (e.g. comparing two values of a user-defined type)

Guidance for developers

- Add support for netCDF enhanced data model features incrementally
  - new primitive types: unsigned numeric types and strings
  - nested Groups (simple recursion)
  - enumeration types (easy, no nesting)
  - opaque types (easy, no nesting)
  - compound types with only primitive members
  - compound types with fixed-size array members
  - variable-length arrays of primitives
  - compound types with members of user-defined type
  - variable-length arrays of user-defined types
- Look at nccopy for examples that read or write netCDF-4 data with all these features

Concluding Remarks

- NetCDF-4's enhanced data model adds representational power
  - Extension of classic model, so maintains compatibility with existing data and programs
  - Adds groups, compound, enumerated, and variable-length types
- Adapting netCDF-3 software to netCDF-4 is practical
  - ncdump, nccopy, ncgen handle all netCDF-4 data model features
  - Incremental adaptation is easy and useful
- Upgrading software to handle features of netCDF-4 enhanced data model has significant benefits
  - Data providers can use more natural representation of complex data semantics
  - More natural conventions become possible
  - End users can access more types of data through netCDF APIs
- Developers offer the best hope for breaking the chicken-and-egg logjam, fighting chained metaphors!

For more information

Web site: www.unidata.ucar.edu/netcdf/

Russ Rew: russ@unidata.ucar.edu

New primitive types

- Unsigned numeric types better for representing data providers intent
  - ubyte: 8-bit unsigned interger
  - ushort: 16-bit unsigned integer
  - uint: 32-bit unsigned integer
- 64-bit integers needed for statistics and counts in large datasets
  - int64: 64-bit signed integer
  - uint64: 64-bit unsigned integer
- Variable-length strings an overdue improvement over character arrays
  - string: compact, variable-length strings

Groups

- Like directories in a file system, Groups provide name spaces and a hierarchy of containers
- Uses
  - Factoring out common information
    - Containers for data within regions, ensembles
    - Model metadata
  - Organizing a large number of variables
  - Providing name spaces for multiple uses of same names for dimensions, variables, attributes
  - Modeling large hierarchies

Variable-length types

Uses:

- Ragged arrays
- Modeling relational tables
- Nested with compound types for in situ observational data (profiles, soundings, time series)
- Example: observations along ocean tracks
  - each track has an ID, a description, and a variable-length list of profiles
    - each profile has a latitude, longitude, time, and a variable-length list of observations
      - each observation records pressure, temperature, and salinity at various depths

Compound types

Uses include:

- Representing vector quantities like wind
- Bundling multiple in situ observations together (profiles, soundings)
- Modeling relational database tuples
- Providing containers for related values of other user-defined types (strings, enums, ...)
- Representing C structures, Fortran derived types portably

Nested types

- Compound types may include other variable-length types or compound types as members
- Variable-length types may include other compound types or variable-length types as members
- Result is a potentially infinite number of user-defined data types
- Handling this in software can be new or intimidating to software developers